Test Set Generation Using Linear Finite State Machines

by

Wanning Tian
B Sc , University of Victoria, 1995

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

in the Department of Computer Science

We accept this thesis as conforming
to the required standard

[redacted]

Dr J C Muzio, Supervisor (Department of Computer Science)

[redacted]

Dr N Horspool, Departmental Member (Department of Computer Science)

[redacted]

Dr G Beer, Outside Member (Department of Physics)

[redacted]

Dr P Driessen, External Examiner (Department of Electrical and Computer Engineering)

Supervisor Dr J C Muzio

Abstract

In this thesis, we investigate the embedding of some deterministic test sets into sequences generated by linear finite state machines The performance of the machines is analyzed using simulation An alternative approach using partitioning is also proposed Our research shows that the embedding is possible However, finding good machines is comparatively difficult We suggest that some modifications to the linear finite state machines may solve the problem We present an approach of partitioning the machines which reduces the length of the test sequence

Examiners

[redacted]

Dr J C Muzio, Supervisor (Department of Computer Science)

[redacted]

Dr N Horspool, Departmental Member (Department of Computer Science)

[redacted]

Dr G Beer, Outside Member (Department of Physics)

[redacted]

Dr P Driessen, External Examiner (Department of Electrical and Computer Engineering)

Table of Contents

List of Tables

List of Figures

Acknowledgments

I wish to give my thanks for my advisor, Dr J C Muzio, for his invaluable encouragement, guidance, and support during my graduate study and thesis work I would also like to thank Kevin Cattell and Bill Gardner for their assistance and suggestions during my thesis work

Chapter 1

Introduction

With the advent of very large scale integration some years ago, the number of transistors integrated into one small piece of silicon approached one million. Also, chips are used in a multitude of different applications of our lives. Consequently, the issues of testing, design-for-test and built-in self-test are becoming increasingly important.

The testing of a circuit requires the set of test stimuli and a method of comparing the response of the circuit with the correct response. Test generation is the process of determining the stimuli necessary to test a digital system. Test generation depends primarily on the testing method employed [3]. There are a number of different approaches to testing. The oldest is to find a specific test set which stimulates each of the individual faults. For a number of reasons, this is not feasible in some modern applications, particularly built-in self-test. In this situation there are two alternatives, either using exhaustive test patterns, that is all possible combinations of the inputs, or some apparently random set of input combinations.

A pseudorandom binary sequence is a string of binary digits such that the bits of the string appear to be random in the local sense and repeat in some way. Linear finite state machines can generate this kind of pseudorandom binary sequence. Hence, we use linear finite state machines as the test generation of these two alternatives. Normally, we use lin-

ear feedback shift registers as the testing generators since they can be easily implemented.

A deterministic test set may get 100% fault coverage. However, for built-in self-test, we need to write the test set into read only memory, and apply each test vector one by one to the circuit under test. Thus, the test speed is low and some extra area of the chips is required. On the other hand, if we use the pseudorandom sequences generated by the linear finite state machines as the test patterns, we have a much large number of test vectors, especially for the circuits which have a large number of inputs. Therefore, if we can combine the advantage of the short sequence length of a deterministic test set with the easily generated property of a pseudorandom sequence, we can improve the testing method so that we have a comparatively high speed, low cost testing method.

The main goal of this thesis is to investigate possible approaches to achieve the above goal. That is, we want to find a linear finite state machine which can generate a pseudorandom sequence such that the test vectors in the deterministic test set appear reasonably close together in the sequence. In order to achieve this goal, we investigate the embedding of some deterministic test sets into sequences generated by linear finite state machines. The simulation results are analyzed and an alternative approach is presented. Finally, we give an implementation of a real deterministic test set to observe how the approach works in practice.

The rest of this introduction gives a summary of the thesis contents. Chapter 2 is the introduction about linear finite state machines. The definitions related to linear finite state machines are given. The theorems and properties related to these machines are formally presented and illustrated with examples. Two typical linear finite state machines, linear feedback shift registers and linear hybrid cellular automata, and their properties are also introduced. This chapter gives readers some general background knowledge.

In chapter 3, the general definitions of m-sequences and their properties illustrated by examples are first presented. Then, an application of the m-sequences is given. Finally, the precise goals of the thesis are introduced. The purpose of this chapter is to give reader the specific knowledge required for the following chapters.

In chapter 4, we do the systematic investigation of the problem using test sets with

small size. In this chapter, we first give an example to illustrate how to use a linear finite state machine to generate the deterministic test set completely. Then, we present an algorithm which can examine whether we can generate each of the possible deterministic test sets by using a linear finite state machine. The simulation results are presented and analyzed. We also deliver some results and theorems related to our problem.

In chapter 5, we do the random investigation of the problem using test sets with large size. We first present an equation which can calculate the total number of linear finite state machines. Then, we use the same algorithm to examine some real deterministic test sets with large size. The simulation results and analysis based on these real test sets in practice are presented. Finally, we give an alternative approach to solving the problem and illustrate this approach by an implementation on a real deterministic test set.

In chapter 6, we give some final conclusions and summarize our main contributions. We also present some discussion about future work.

Chapter 2

Linear Finite State Machines

Many linear finite state machines can be used to generate pseudorandom sequences. Linear feedback shift registers and linear cellular automata are commonly used. Since these pseudorandom sequences are repeatable, they have many interesting properties.

2.1 Linear Finite State Machines

2.1.1 Mathematical Definitions

We need to know some mathematical definitions and theorems to understand the later topics. The following is a set of some algebraic definitions and theorems which we required, and which are related to linear finite state machines. The main sources for this section are [1, 6].

Definition 2.1 A *finite-state machine* is an algebraic structure $\langle S, I, Y, M, \delta \rangle$, where S, I, and Y are finite sets of states, inputs, and outputs, respectively, M is a mapping from $S \times I$ into S, and δ is a mapping from S into Y.

Figure 2.1 shows a finite-state machine. The machine model we use here is called the

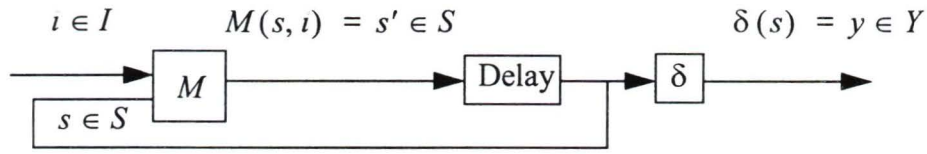

Figure 2.1 An abstract model of a finite-state machine

Moore model. The Mealy model of finite-state machines is an alternative model. For the Mealy model, the δ (output function) is a mapping from $S \times I$ into Y. We can specify the behavior of a finite-state machine by specifying its next-state and output functions. For example, the table and graph in Figure 2.2 describe a finite-state machine that produces a 1 output when the preceding four inputs form the sequence 0100.

Definition 2.2 An *Abelian group* G is a set of elements which satisfy

1. The sum of any two elements of G is still in G.
2. For any elements a, b, c in G, $a + b = b + a$, and
   $(a + b) + c = a + (b + c)$
3. There is an element 0 which satisfies $a + 0 = a$ for every a in G.
4. Every element a has a negative $\bar{a}$ such that $a + \bar{a} = 0$.

Definition 2.3 A *field* F is a set of elements with two operations + (called addition) and $\cdot$ (called multiplication) such that

1. it is an Abelian group under + with identity 0,
2. the non-zero elements form an Abelian group under $\cdot$ with identity 1 $(1 \neq 0)$,
3. the distributive law holds, that is, $x \cdot (y + z) = (x \cdot y) + (x \cdot z)$ and
   $(y + z) \cdot x = (y \cdot x) + (z \cdot x)$ for all x, y, z in F.

For example, $\langle Q, +, \cdot \rangle$ and $\langle R, +, \cdot \rangle$ are fields (Q is the rationals, R the reals).

Definition 2.4 A *finite field* is a field which has a finite number of elements. The number of elements in the field is called the order of the field.

For example, $GF(p)$, the *Galois field of order* p, is a finite field.

Present State S	$NextState$ $M(s, \iota)$		Output $\delta(s)$
	$\iota = 0$	$\iota = 1$	
s_1	s_2	s_1	0
s_2	s_2	s_3	0
s_3	s_4	s_1	0
s_4	s_5	s_3	0
s_5	s_2	s_3	1

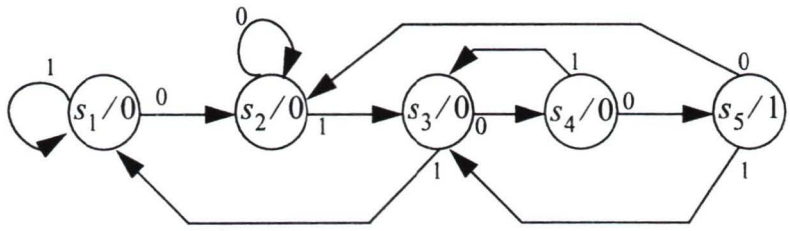

Figure 2 2 State table and graph for the next-state and output functions of a machine M

Definition 2 5 V is a *vector space* over K if $\langle V, K, +, \rangle$ is a unitary module and K is a field

Definition 2.6 A polynomial $p(x)$ of degree $n > 0$ over K is *irreducible* over K if and only if there do not exist polynomials $f(x)$ and $g(x)$ of degree greater than 0 over K such that $f(x)g(x) = p(x)$, where multiplication is ordinary polynomial multiplication with coefficient operations in K

$x^3 + x + 1$, $x^4 + x + 1$, and $x^4 + x^3 + 1$ are all irreducible polynomials over $GF(2)$, $x^3 + 2x + 1$, $x^4 + x + 2$, and $x^4 + 2x^3 + 2$ are all irreducible polynomials over $GF(3)$

Definition 2 7 A polynomial $p(x)$ of degree n is a *monic* polynomial if the coefficient of x^n is unity

These definitions provide some of the mathematical background which is helpful to our understanding of linear finite state machines In the following chapters, our field is

$GF(2)$ unless otherwise stated

2.1.2 General Definitions

A linear finite state machine has many useful and interesting properties They have been extensively studied We give some basic definitions, theorems, and examples of linear finite state machines here The main source of this section is [1]

Definition 2.8 A machine M is a *linear finite-state machine* if

1 the state space S_M of M, the input space I_M, and the output space Y_M are each vector spaces over a finite field K (we let the dimensions of the spaces be n, p, and r, respectively), and

2 for the state vector s_t, input vector u_t, and output vector y_t at time t, the next-state and output functions are of the forms

$$M(s_t, u_t) = s_{t+1} = A\ s_t + B\ u_t$$

and $\delta(s_t) = y_t = C\ s_t$,where A, B, and C are matrices over K, s_t, u_t, and y_t are column vectors, and the matrix operations are the usual matrix operations with arithmetic performed in the field K

By using the above two equations, we can calculate the next state and the output for a particular linear finite state machine For example, let M be a linear finite state machine with the matrices A, B, and C such that

$$A = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix} \qquad B = \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 1 \end{bmatrix} \qquad C = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Assume that M is started in state $\langle 0, 1, 0 \rangle^t$ and the input is $\langle 1, 1 \rangle^t$ From the above assumption, we can get the next state s_{t+1} as

$$s_{t+1} = A\ s_t + B\ u_t = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix} \langle 0, 1, 0 \rangle + \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 1 \end{bmatrix} \langle 1, 1 \rangle = \langle 1, 0, 0 \rangle$$

and the output y_t is

$$y_t = C\ s_t = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix} \langle 0, 1, 0 \rangle = \langle 1, 1 \rangle$$

Definition 2.9 The state $s = (0, 0, \ldots, 0)^t$ of a linear machine is called the *0-state*, or *state 0*, and the input $u = (0, 0, \ldots, 0)^t$ is called the *0-input*, or *input 0*.

Theorem 2.1 [1] For every input sequence $u_1 u_2 \ldots u_m$ and for every state s_1,

$$M(s_1, u_1 u_2 \ldots u_m) = M(s_1, 0^m) + \sum_{i=1}^{m} M(0, v_i)$$

where each v_i is a sequence of 0-inputs except for input u_i at time i and where M is a linear next-state function.

This theorem means that the behavior of M on a sequence of inputs is equal to the superposition of its behavior for individual inputs. We can use matrix notation to rewrite the above equation [1]

$$M(s_1, u_1 u_2 \ldots u_m) = A^m\ s + \sum_{i=1}^{m} A^{m-i}\ B\ u_i$$

2.2 Autonomous Linear Finite State Machines

Definition 2.10 A linear finite state machine which has no inputs is called an *autonomous linear finite state machine*.

In Figure 2.3 we give the block diagram of an autonomous linear finite state machine. So the next state function is $s_t^+ = s_t\ T$, where T is the transition matrix. In this thesis, when

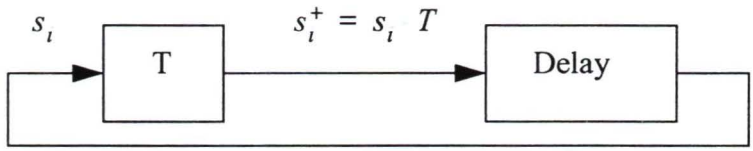

Figure 2.3 Block diagram of an autonomous linear finite state machine

ever an LFSM (linear finite state machine) is used, it means an autonomous linear finite state machine, unless otherwise stated.

Definition 2.11 A linear finite state machine is said to have a *maximum length cycle* if all possible states except the 0-state lie on a single connected cycle in the state transition graph.

Consider an example here, with two transition matrices T_1 and T_2

$$T_1 = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix} \qquad T_2 = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix}$$

And the state transition graphs are as shown in Figure 2.4 and 2.5 (Note that the decimal labelling of the states is an abbreviation for the corresponding bit pattern.)

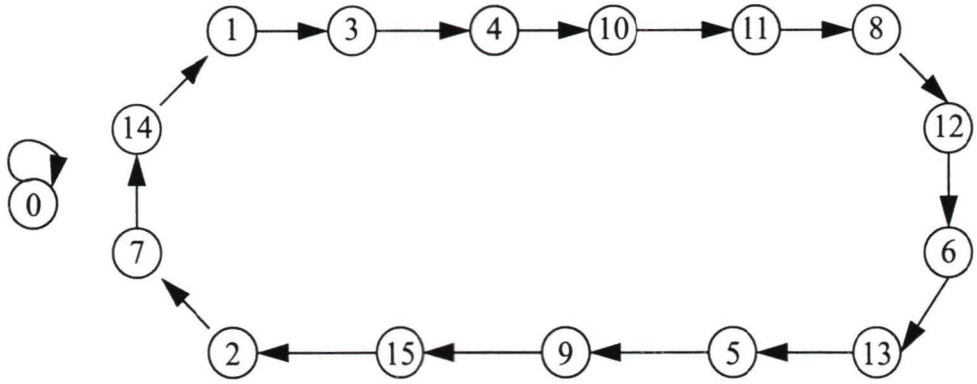

Figure 2.4 State transition graph of LFSM with T_1

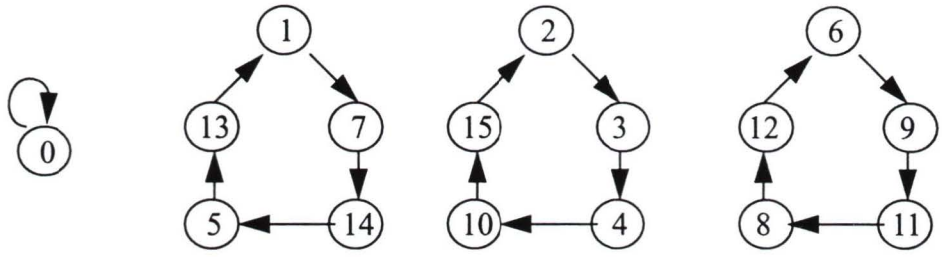

Figure 2 5 State transition graph of LFSM with T_2

Figure 2 4 gives a LFSM which has a maximum length cycle and Figure 2 5 gives a LFSM which does not have a maximum length cycle

Definition 2 12 An $n \times n$ matrix A is said to be *nonsingular* if $det(A) \neq 0$, where the determinant is calculated in the arithmetic of the field K

We can easily see that $A = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}$ is nonsingular since $det(A) = 1$ and that

$B = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 0 & 1 \end{bmatrix}$ is singular since $det(B) = 0$

Definition 2 13 If A is any $n \times n$ matrix, and if Q is a nonsingular $n \times n$ matrix, then the matrix $A' = Q\ A\ Q^{-1}$ is said to be *similar* to A

Let A be $\begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 1 & 1 & 0 \end{bmatrix}$, Q be $\begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}$ Q is nonsingular since $det(Q) \neq 0$ So Q^{-1} is

$\begin{bmatrix} 1 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}$ And we get

$$A' = Q\ A\ Q^{-1} = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

It is very difficult to determine if two matrices are similar since it is not easy to find the Q. One way we can do it is to find some X_1, X_2 and λ_1, λ_2 from

$$A_1X_1 = \lambda_1X_1 \qquad A_2X_2 = \lambda_2X_2$$

if $X_1 = X_2$ and $\lambda_1 = \lambda_2$, then A_1 and A_2 are similar.

Theorem 2.2 [1] If matrices T and T' are similar nonsingular state-transition matrices, then their state graphs have identical cycle structures and differ only in the labeling of the states.

From the above example, the state graphs for $T = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 1 & 1 & 0 \end{bmatrix}$ and $T' = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}$ are shown in Figure 2.6. We can apply the theorem to this example and conclude that similar nonsingular state-transition matrices give the same structure of the state graphs with different order (or labeling) of the states.

Definition 2.14 The *characteristic polynomial* $\phi_A(\lambda)$ of an $n \times n$ matrix A is the polynomial $det(\lambda I - A)$.

We get the characteristic polynomial for $A = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}$ by using the definition. The characteristic polynomial is

$$\Phi_A(\lambda) = \lambda^3 + \lambda + 1$$

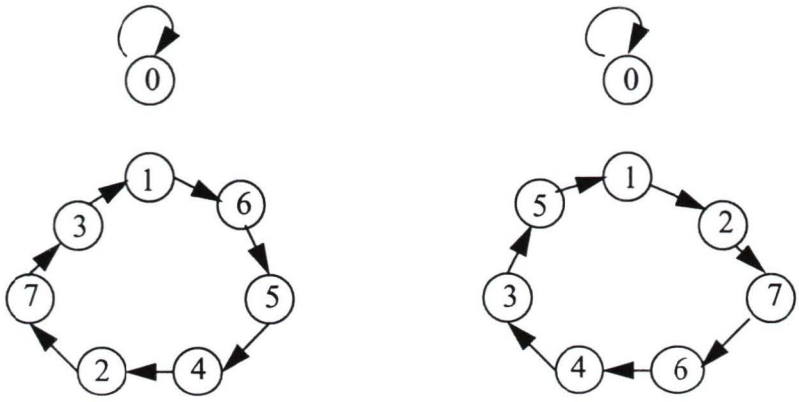

Figure 2 6 State graphs for LFSMs with transition matrices T and T'

From the theorem 2 2 we can see that similar state transition matrices produce the same cycle structure This means we can obtain a particular cycle structure by constructing any machine in an equivalence class characterized by that structure, and we can select the machine of least cost in an equivalence class This is the reason for us to look at linear feedback shift registers and linear hybrid cellular automata

2.3 Linear Feedback Shift Registers

A linear feedback shift register (LFSR) is an LFSM defined as a collection of memory cells and XOR gates which are chained together and controlled by a synchronous clock There are two configurations for the LFSR a Type I LFSR which has the XOR gates between the cells, and a Type II LFSR which has XOR gates on the feedback path Each LFSR has its own state transition matrix The generation function $f(x)$ of each LFSR is called the characteristic polynomial We denote a type I LFSR by LFSR(I) and a type II LFSR by LFSR(II) Figure 2 7 and Figure 2 8 show the typical LFSR(I) and LFSR(II) Here c_i is a binary constant, and $c_i = 1$ implies that a connection exists, while $c_i = 0$

implies that no connection exists

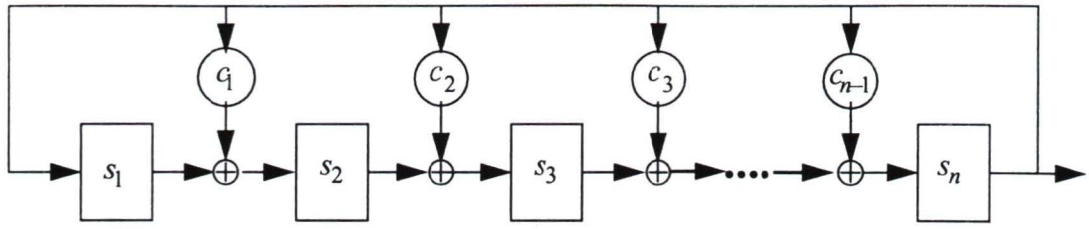

Figure 2.7 A type I linear feedback shift register with characteristic polynomial $P(x) = x^n + c_{n-1}x^{n-1} + \ldots + c_2x^2 + c_1x + 1$

From Figure 2.7 we can get the relationship between the current state and the next state. Let the current state be $s = \langle s_1, s_2, s_3, \ldots, s_n \rangle$ and the next state be $s^+ = \langle s_1^+, s_2^+, s_3^+, \ldots, s_n^+ \rangle$

Then

$$\begin{cases} s_1^+ = s_n \\ s_i^+ = s_{i-1} \oplus c_{i-1}s_n \qquad for\ i = 2, 3, \ldots, n \end{cases}$$

So $s^+ = s \cdot T$ and the transition matrix T is

$$T = \begin{bmatrix} 0 & 1 & 0 & \ldots & 0 \\ 0 & 0 & 1 & \ldots & 0 \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ 0 & 0 & 0 & \ldots & 1 \\ 1 & c_1 & c_2 & \ldots & c_{n-1} \end{bmatrix} \qquad (2.1)$$

Alternatively for type II, from Figure 2.8 we can get

$$\begin{cases} s_1^+ = \sum_{i=1}^{n-1} c_i s_{n-i} \\ s_i^+ = s_{i-1} \qquad for\ i = 2, 3, \ldots, n \end{cases}$$

and the transition matrix T for LFSR(II)

$$T = \begin{bmatrix} c_{n-1} & 1 & 0 & \dots & 0 \\ c_{n-2} & 0 & 1 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ c_1 & 0 & 0 & \dots & 1 \\ 1 & 0 & 0 & \dots & 0 \end{bmatrix} \qquad (2.2)$$

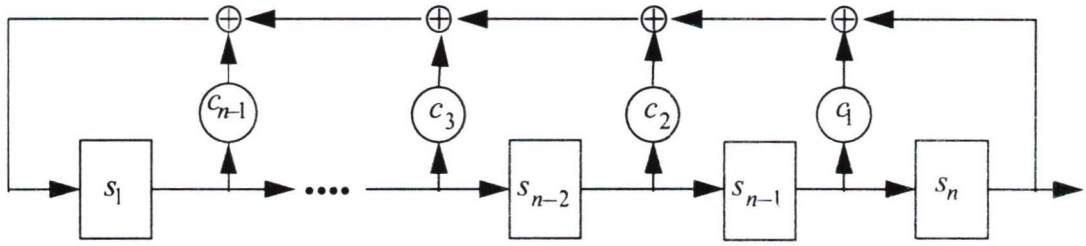

Figure 2.8 A type II linear feedback shift register with characteristic polynomial $P(x) = x^n + c_{n-1}x^{n-1} + \dots + c_2x^2 + c_1x + 1$

Figure 2.9 shows the two types of LFSRs derived from the same polynomial $x^5 + x^3 + 1$ [4]

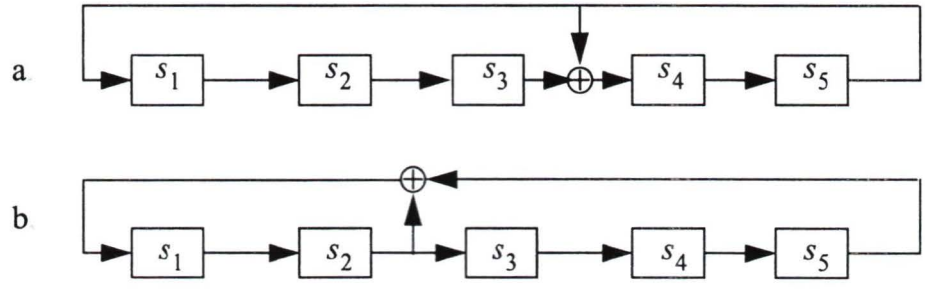

Figure 2.9 a Type I LFSR and b Type II LFSR with characteristic polynomial $x^5 + x^3 + 1$

By using the transition matrices (2.1) and (2.2) we can get the corresponding state

transition matrices of the LFSRs in Figure 2.5, namely

$$(a) \quad T = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \end{bmatrix} \qquad (b) \quad T = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

In the following section, we introduce an alternative interesting linear finite state machine: linear hybrid cellular automata.

2.4 Linear Hybrid Cellular Automata

The linear hybrid cellular automata (LHCA) is defined as a uniform array of identical cells in a one-dimensional space. Cells are restricted to local neighborhood interaction and have no global communication. The algorithm for cells to compute their next states is called the computation rule. Figure 2.10 is an example of a LHCA with Rule 90 and 150. It is said to have null boundary conditions when the two end inputs always have 0 value (See [8] for a detailed discussion of LHCA.)

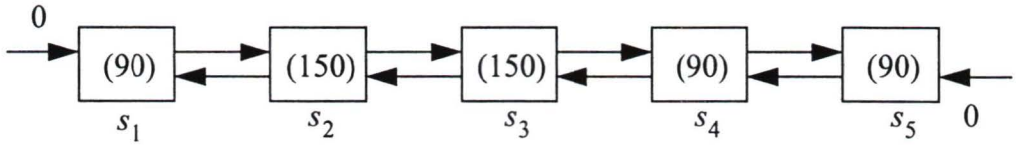

Figure 2.10 An LHCA with characteristic polynomial

$$P(x) = x^5 + x^3 + 1$$

Definition 2.15 Let s_i, $1 \le i \le n$, be the current state of cell i of the LHCA identified by the n-tuple $(c_1, c_2, \ldots, c_n)$. Its next state, s_i^+, is given by $s_i^+ = s_{i-1} \oplus c_i s_i \oplus s_{i+1}$, where $s_0 = s_{n+1} = 0$, $c_i = 0$ for a Rule 90 cell and $c_i = 1$ for a Rule 150 cell.

We only consider LHCA with Rule 90 and Rule 150 cells in this thesis since it has

been shown in [8] that this is a requirement for the LHCA to have a maximum length cycle A general state transition matrix for the LHCA is

$$T = \begin{bmatrix} c_1 & 1 & 0 & 0 & 0 \\ 1 & c_2 & 1 & 0 & 0 \\ & & \cdots & & \\ 0 & & 1 & c_{n-1} & 1 \\ 0 & & 0 & 1 & c_n \end{bmatrix} \qquad (2.3)$$

By using (2 3), we can get the transition matrix for LHCA of Figure 2 10

$$T = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Definition 2.16 [8] For an n-cell LHCA $(c_1, c_2, \ldots, c_n)$, its *characteristic polynomial* is given by the function defined recursively as follows

$$\begin{cases} \Delta_n(x) = (x + c_n)\Delta_{n-1}(x) + \Delta_{n-2}(x) \\ \Delta_1(x) = x + c_1 \\ \Delta_0(x) = 1 \end{cases} \qquad (2.4)$$

By using (2 4), we can calculate the characteristic polynomial for LHCA of Figure 2 10 as follows

$$\begin{aligned} \Delta_0(x) &= 1 \\ \Delta_1(x) &= x + c_1 = x + 0 = x \\ \Delta_2(x) &= (x + c_2)(x) + 1 = x^2 + x + 1 \\ \Delta_3(x) &= (x + c_3)\left(x^2 + x + 1\right) + (x) = x^3 + x + 1 \\ \Delta_4(x) &= (x + c_4)\left(x^3 + x + 1\right) + \left(x^2 + x + 1\right) = x^4 + 1 \\ \Delta_5(x) &= (x + c_5)\left(x^4 + 1\right) + \left(x^3 + x + 1\right) = x^5 + x^3 + 1 \end{aligned}$$

Thus the characteristic polynomial for LHCA of Figure 2.10 is $P(x) = x^5 + x^3 + 1$. Note that we normally use the diagonal of the transition matrix to represent the LHCA. We call it the rule of the LHCA.

We are more interested in the LFSMs which can generate a sequence including all nonzero states. The next chapter describes the definitions and theorems related to this topic.

Chapter 3

M-Sequences

For an n-stage LFSM, it can have at most 2^n possible different states. Since it is linear, any successor of the 0-state will be 0-state. So, starting with a nonzero state, the n-stage LFSM can reach at most $2^n - 1$ different states. That is, an LFSM can generate a sequence periodically and has a maximum value of $2^n - 1$. Also the sequence produced is a pseudo-random sequence since it is a string of digits such that the digits appear to be random in the local sense and repeat in some way. These pseudorandom sequences with period of $2^n - 1$ are very useful in many areas, such as ranging systems, secure communication systems and digital computer systems. Therefore, we are very interested in these sequences which are called m-sequences. In this chapter, we introduce the general definitions and theorems related to m-sequences and some applications using m-sequences.

3.1 Mathematical Definitions

In order to introduce the m-sequences, we need some basic mathematical concepts. The following are some definitions and theorems which help us understand more about m-sequences.

Definition 3.1 [2] For any monic polynomial $f(x)$, the *period* of $f(x)$ is defined to be the least integer k such that $f(x)$ divides $x^k + 1$.

For example $x^4 + x + 1$ has period of 15 since 15 is the least integer such that $x^{15} + 1$ can be divided by $x^4 + x + 1$, and $x^4 + x^3 + x^2 + x + 1$ has period of 5 since 5 is the least integer such that $x^5 + 1$ can be divided by $x^4 + x + 1$. We have two theorems about the *period*.

Let the sequence $\{a_n\} = \{a_0, a_1, \ldots\}$ represent the history of the output stage of an LFSM where $a_t = 1$ or 0 depending on the state of the output stage at time t_t. Then we have the following theorem.

Theorem 3.1 [2] If a sequence $\{a_n\}$ is derived from an n-stage LFSR with irreducible characteristic polynomial $P(x)$, then the period of the sequence $\{a_n\}$ is a factor of $2^n - 1$.

Theorem 3.2 [1] Let $P(x)$ be the irreducible characteristic polynomial of a shift register with next-state matrix T. Then, if k is the period of $P(x)$, all nonzero states lie on cycles of length k.

For example, an LFSR with irreducible characteristic polynomial $x^4 + x^3 + x^2 + x + 1$ has the period of 5 by definition and 5 is a factor of 15. And its state transition graph is shown in Figure 3.1.

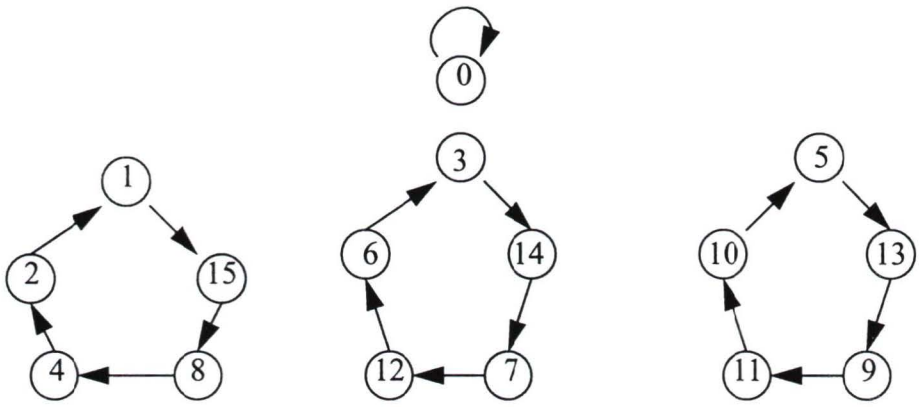

Figure 3.1 State graph for the LFSR with characteristic polynomial $x^4 + x^3 + x^2 + x + 1$

We see it is correct that all the nonzero states lie on cycles of length 5.

Definition 3.2 [1] An element α of the field $GF(q)$ is called a *primitive element* if it generates the multiplicative group of the field.

Definition 3.3 [2] The *reciprocal polynomial* $f^*(x)$ of $f(x)$ is $f^*(x) = x^n f(1/x)$. The polynomial $x^3 + x + 1$ is the reciprocal polynomial of $x^3 + x^2 + 1$.

Definition 3.4 [2] If the output sequence $\{a_n\}$ generated by an n-stage LFSM has period $2^n - 1$, it is called a *maximum-length sequence* or *m-sequence*. The characteristic polynomial of the LFSM that produces a maximum-length sequence is called a *primitive polynomial*.

Polynomials such as $x^3 + x + 1$, $x^4 + x + 1$ and $x^5 + x^2 + 1$ are primitive polynomials.

The number of primitive polynomials for an n-stage LFSM is given by the formula [6]

$$k(n) = \Phi\left(2^n - 1\right)/n$$

where $\Phi(n) = n\sum_{p|n}(1 - 1/p)$ and p is taken over all primes that divide n. Table 3.1

shows some values of $k(n)$

n	k(n)
1	1
2	1
4	2
8	16
16	2048
32	67108864

Table 3.1 The number of primitive polynomials for degree n

In the next section, we describe some properties of m-sequences.

3.2 Properties of M-Sequences

M-sequences have many interesting properties and are very useful in various areas of digital systems. The following are some of the important properties and their proofs of m-sequences which we require for our later work [2]. Suppose we are given an m-sequence $\{a_n\}$ with characteristic polynomial $P(x)$, a primitive polynomial of degree n, then

Property I The period of $\{a_n\}$ is $p = 2^n - 1$, that is, $a_{p+i} = a_i$ for all $i \geq 0$.

This property comes directly from the definition.

Property II Starting from a nonzero state, the LFSM that generates $\{a_n\}$ goes through all $2^n - 1$ states before repeating.

This property is obvious since the period of the m-sequence is $2^n - 1$.

Property III The number of 1's in an m-sequence differs from the number of 0's by one.

Proof Since the n-stage LFSMs which generate the sequence cycles through all $2^n - 1$ nonzero states (from Property II), it can be thought of as displaying all the numbers from 1

to $2^n - 1$. And the m-sequence can be thought of as the first bits of all those numbers. Since from 1 to $2^n - 1$ there are 2^{n-1} odd numbers and $2^{n-1} - 1$ even numbers, therefore, there are 2^{n-1} 1's and $2^{n-1} - 1$ 0's and the disparity is one.

Q E D

Consider an example with $f(x) = x^3 + x + 1$, we have the m-sequence 0101110 which has four 1's and three 0's. Below, we use the same m-sequence as an example for the following properties.

Property IV (The Window Property) If a window of width n is slid along an m-sequence, then each of the 2^{n-1} nonzero binary n-tuples is seen exactly once in a period. For our example and a window of width 3, this gives 010, 101, 011, 111, 110, 100, 001. That is each of the 7 nonzero binary 3-tuples is seen exactly once in a period of 7.

The concept of a run is introduced. A maximal contiguous grouping of symbols is called a "run." To avoid ambiguity in periodic sequences, consider the sequence written in a circle to avoid end effects. To count maximal contiguous groupings, the period must start at the transition from a run of 1's to a run of 0's or vice versa.

Property V (The Run Property) In every period of an m-sequence, one-half of the runs have length 1, one-fourth have length 2, one-eighth have length 3, and so forth, as long as the fractions yield integral numbers of runs. The runs of 1's and 0's terminate with runs of length n and $n-1$, respectively. Moreover, except for these terminating lengths, there are equally many runs of 1's and of 0's. The total number of runs of 1's equals the total number of runs of 0's. The number of runs in a period is $(m+1)/2$, where $m = 2^n - 1$.

Proof By Property IV, we know that n consecutive 1's occurs exactly once. This run must be preceded by a 0 and followed by a 0. Since all the n-tuples are seen exactly once by Property IV, the n consecutive 1's preceded by a 0 and followed by a 0 includes the run of $n-1$ consecutive 1's preceded by a 0 and followed by a 0.

There is no run of n consecutive 0's since we do not include the all 0-state here. So a 1

followed by $n-1$ consecutive 0's must occur, so there is a run of $n-1$ consecutive 0's.

Now, let $0 < k < n-1$. To find the number of runs of 1's of length k, consider n consecutive bits beginning with 0, then k 1's, then a 0, and then the remaining $n-k-2$ bits arbitrary. This occurs in 2^{n-k-2} ways since each n-tuple occurs exactly once. Therefore, there are 2^{n-k-2} runs of k consecutive 1's. A similar argument is true for the runs of 0's. Hence, we can get

$$runs\,(1\text{'s}) = 1 + \sum_{k=1}^{n-2} 2^{n-k-2} = 2^{n-2}$$

$$runs\,(0\text{'s}) = 1 + \sum_{k=1}^{n-2} 2^{n-k-2} = 2^{n-2}$$

So the total number of runs is $2 \times 2^{n-2} = 2^{n-1}$. Since $m = 2^n - 1$, and $(m+1)/2 = 2^{n-1}$, the number of runs.

The number of runs of 1's of length $k = 1$ is $2^{n-k-2} = 2^{n-3}$ and the number of runs of 1's of length $k = 2$ is $2^{n-k-2} = 2^{n-4}$. Hence, one-half the runs of 1's have length 1, one-fourth have length 2, and so on. We have the similar statement for the runs of 0's.

Q E D

For our example, m-sequence 0101110, the number of runs should be $(m+1)/2$ where $m = 7$, i e 4. We actually have runs 0, 1, 00, 111 and $2 = \frac{1}{2} \times 4$ of the runs have length 1, $1 = \frac{1}{4} \times 4$ of the runs has length 2, and $1 = \left\lceil \frac{1}{8} \times 4 \right\rceil$ of the runs has length 3. The runs of 1's terminate with runs of length 3, and the runs of 0's terminate with runs of length 2.

Property VI (Transitions) The number of transitions between 1 and 0 that an m-sequence makes in one period is $(m+1)/2$, where $m = 2^n - 1$.

Proof Since a transition is associated with the beginning of a run, and the number of runs is $(m+1)/2$, so the number of transitions is $(m+1)/2$

Q E D

For our example m-sequence, 0101110, we have

$$0 \rightarrow 1,\ 1 \rightarrow 0,\ 0 \rightarrow 1,\ notransition,\ notransition,\ 1 \rightarrow 0,\ notransition$$

The number of transitions between 1 and 0 in one period is $(7+1)/2 = 4$

Property VII (The Shift-and-Add property) The sum of any m-sequence A_t and a cyclic shift of itself is another cyclic shift of A_t

Proof Let $A_1 = \{a_1, a_2, a_3, \ldots\}$ be the m-sequence $\{a_n\}$ Let $A_2 = \{a_2, a_3, a_4, \ldots\}$, $A_3 = \{a_3, a_4, a_5, \ldots\}$, $\ldots$, $A_p = \{a_p, a_1, a_2, \ldots\}$, where $p = 2^n - 1$ Also, let $A_0 = \{0, 0, 0, \ldots\}$ Let R be the linear recurrence relation satisfied by A_1 Then R is also satisfied by $A_2, A_3, \ldots, A_p$ and A_0 Since R is linear, it is also satisfied by $A_t + A_j$ where $A_t + A_j$ denotes the term-by-term sum Thus $A_t + A_j$ is determined by R from its first n terms And these n terms are the same as the first n terms of exactly one of the 2^n sequences $A_0, A_1, \ldots, A_p$ Thus the sum of two cyclic shifts of an m-sequence is a cyclic shift of the same m-sequence

Q E D

Using our example, m-sequence 0101110, we can let A_t = 0101110 and a cyclic shift of A_t be 1110010 So the sum of them is 0101110 + 1110010 = 1011100, which is another cyclic shift of A_t

In order to define the autocorrelation function, the sequence $\{a_n\}$ with a_n belonging to $\{0, 1\}$, is changed into an equivalent sequence $\{b_n\}$ by defining $b_t = 1 - 2a_t$, where

the minus sign indicates an arithmetic operation. Thus 1's in $\{a_n\}$ are replaced by -1's in $\{b_n\}$ and the 0's by +1's. For a periodic sequence such as a shift-register sequence, the autocorrelation function is defined as

$$C(\tau) = \frac{1}{p}\sum_{n=1}^{p} b_n b_{n-\tau}$$

Property VIII The autocorrelation function of every m-sequence of period $p = 2^n - 1$ is given by

$$C(0) = 1$$

$$C(\tau) = -1/p \qquad for \qquad 1 \le \tau \le 2^n - 2$$

For our example, m-sequence 0101110, then $\{a_n\}$ is

$$0, 1, 0, 1, 1, 1, 0$$

and $\{b_n\}$ is

$$+1, -1, +1, -1, -1, -1, +1$$

So by definition

$$C(0) = \frac{1}{7}\sum_{n=1}^{7} b_n b_n = \frac{1}{7}(b_1 b_1 + b_2 b_2 + \ldots + b_7 b_7) = \frac{1}{7}\ 7 = 1$$

and

$$C(1) = \frac{1}{7}\sum_{n=1}^{7} b_n b_{n-1} = \frac{1}{7}\ (-1) = -1/7$$

and also

$$C(2) = C(3) = C(4) = C(5) = C(6) = C(7) = -1/7$$

Decimation is the selection of every qth element of the sequence. If q is relatively prime to the period $p = 2^n - 1$, the decimation is called proper.

Property IX (Decimation) Every proper decimation of an m-sequence is itself an m-sequence.

Again, using our m-sequence 0101110, we can take $q = 2$ and the resulting sequence is

$$1, 1, 1, 0, 0, 1, 0$$

which is again an m-sequence.

We now have our necessary basic knowledge about m-sequences. Since the number of primitive polynomials is a constant for each degree, the number of m-sequences is a constant too.

All m-sequences repeat in every period. In general, we use one period of the m-sequence to represent the m-sequence. Table 3.2 shows the m-sequences for some of the primitive polynomials with degree less than 7. We get the primitive polynomials from [5].

3.3 An Application of M-sequences

In a communication system, we are always concerned about security. Using the m-sequences, we can get a secure communication system [1]. Figure 3.2 illustrates an insecure communication system.

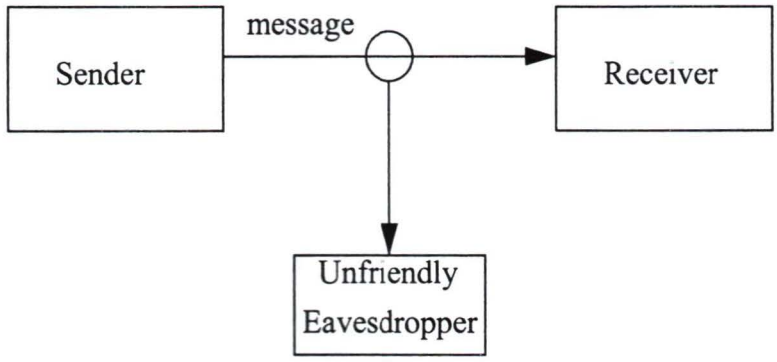

Figure 3.2 A communication system

Primitive Polynomial	M-Sequence
$x^2 + x + 1$	011
$x^3 + x + 1$	0101110
$x^3 + x^2 + 1$ *	0111010
$x^4 + x + 1$	010011010111100
$x^4 + x^3 + 1$ *	011110101100100
$x^5 + x^2 + 1$	010010110011111000110111010100 0
$x^5 + x^3 + 1$ *	010101110110001111100110100100 0
$x^5 + x^4 + x^2 + x + 1$	011100110111110100010010101100 0
$x^5 + x^4 + x^3 + x + 1$ *	011010100100010111110110011100 0
$x^5 + x^3 + x^2 + x + 1$	010110101000111011111001001100 0
$x^5 + x^4 + x^3 + x^2 + 1$ *	011001001111101110001010110100 0
$x^6 + x^5 + 1$	000001111110101011001101110110100100111000 1 011110010100011000 01
$x^6 + x + 1$ *	000001000011000101001111010001110010010110 1 110110011010111111
$x^6 + x^5 + x^2 + x + 1$	000001111001001010100110100001000101101111 1 101011100011001110 11
$x^6 + x^5 + x^4 + x + 1$ *	000001101110011000111010111111011010000100 0 101100101010010011 11
$x^6 + x^4 + x^3 + x + 1$	000001011111100101010001100111101110101101 0 011011000100100001 11
$x^6 + x^5 + x^3 + x^2 + 1$ *	000001110000100100011011001011010111011110 0 110001010100111111 01

Table 3 2 The m-sequences for all the primitive polynomials with degree less than 7

Note * Each such polynomial is the reciprocal polynomial of the entry immediately preceding it The m-sequences are the reverses of each other

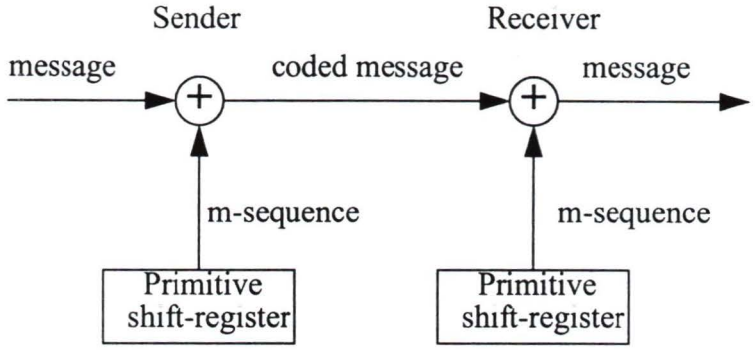

Figure 3 3 A secure communication system based on a simple model of primitive LFSRs

We want to find a simple cryptographic system to encode the message between the sender and receiver, so that the eavesdropper can not understand what he or she hears. Figure 3 3 shows our simple model for the encoder and decoder.

In the sender part, assume the information digit at time t is q_t, and the m-sequence digit at time t is p_t, we get the transmitted digit at time t by $v_t = q_t + p_t$, where v_t is the transmitted digit at time t. In the receiver part, we repeat the same process using the same primitive shift-register. Since the field is $GF(2)$, we get $q_t = v_t + p_t$, where v_t is what we receive at time t, and p_t is the m-sequence digit at time t.

Since the m-sequence has certain pseudo-randomness properties, the coded message will appear to be random and it is hard for the eavesdropper to understand the message. However, we need to synchronize the registers in the encoder part and the decoder part. Many techniques can be used to solve this problem.

We give a simple example to conclude this section. Assume we have an m-sequence 0101110010111..., and our message is 11100100101. So after encoding, we have

$$10111000000 = 11100100101 + 01011100101$$

and in the receiver part, after we receive 10111000000, we can decode the message

$$10111000000 + 01011100101 = 11100100101$$

At the same time, it is hard for the eavesdropper to understand the coded message

There are many applications of m-sequences More detailed material can be found in Knuth [9], Shannon [10], and Golomb [11] about these applications

3.4 A Particular Problem

The sequences generated by the LFSMs, for example LFSRs or LHCA, can be used in circuit testing because of their pseudo-randomness properties Figure 3 4 shows the appropriate architecture for testing

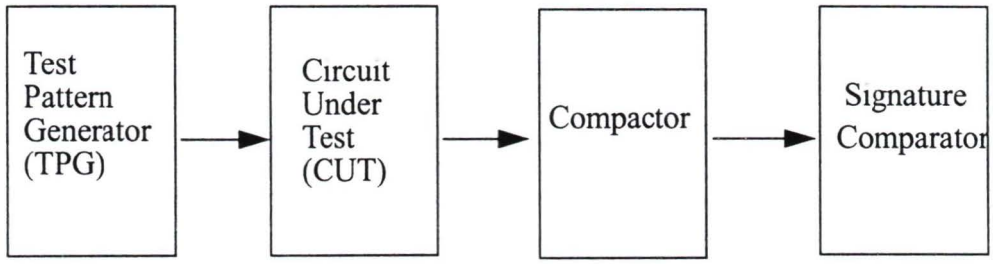

Figure 3 4 The architecture of the testing

In this thesis, we are only concerned with the TPG (Test Pattern Generator) In Chapter one, We introduced two types of random test pattern generator One uses patterns generated by the LFSMs, and the other is a set of test vectors written in ROMs We show an example here Let our circuit to be tested be that shown in Figure 3 5

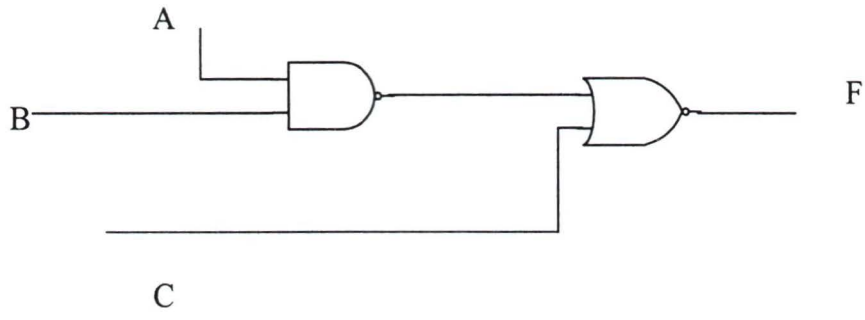

Figure 3 5 A 3-Input circuit with $F = \overline{\overline{AB} + C}$

We get the truth table for this circuit first, as shown in Table 3 3

A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

Table 3 3 The truth table of the circuit in Figure 3 5

Now we can simulate the combination of inputs in two ways one is by using an LFSM as the test pattern generator to simulate all the combinations except all 0-inputs, the other way is by choosing some of the combinations as a set of test vectors

If we decide to use the first method, we notice that we have three inputs, so we know we have to have an LFSM with a characteristic polynomial of degree three And this polynomial must be primitive, so that we can get the maximum length cycle We only have two primitive polynomials $x^3 + x + 1$ and $x^3 + x^2 + 1$ Both of them can give us the maximum length cycle Suppose we choose a 3-stage LFSR whose characteristic polynomial is $x^3 + x^2 + 1$ to simulate the inputs (Figure 3 6) We connect A to the output of s_1, B to s_2 and C to s_3

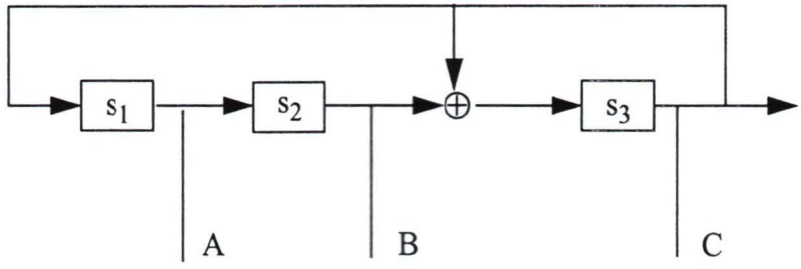

Figure 3 6 A test pattern generator--an LFSR with characteristic polynomial $x^3 + x^2 + 1$

And for each time t the states of ALFSR are shown in Table 3.4

Time	s_1	s_2	s_3	F
t_1	0	0	1	0
t_2	1	0	1	0
t_3	1	1	1	0
t_4	1	1	0	1
t_5	0	1	1	0
t_6	1	0	0	0
t_7	0	1	0	0
	A	B	C	

Table 3.4 The state patterns of the machine in Figure 3.6

After seven cycles, we have simulated all the combinations of inputs except the all-0 inputs.

If we use the second method, i.e. using a deterministic test set, for example, we test the single stuck-at faults by using the test set {111, 110, 100, 010} as the set of test patterns. The easy way to get these patterns is to write this test set into ROM and apply it to our circuit. But this can be expensive.

Since the patterns generated by an LFSM with a primitive polynomial generating all the combinations except 0-state, we can guarantee that the patterns include any test set which does not include the all-0 vector. It seems that the second method is not necessary. So why do we still keep the second method? The answer is because the number of inputs may be very large in the circuit to be tested. For example, if the circuit has 16 inputs, then all combinations of these inputs give 2^{16} combination. And we may only need 10 of these vectors to be our test patterns. Therefore, if we use an LFSM as the test pattern generator here, we may waste time.

Is there a way to combine these two methods? Let us go back to our problem. By observation, we notice that if we start our LFSR with the state vector 111, and after 4 cycles, we have a sequence {111, 110, 011, 100, 010}. Therefore, we include our test set

in a short sequence In this case, we only waste one state vector So, if we are given an n-bits deterministic test set with m vectors in it, we want to find an LFSM which can generate a sequence starting with a vector in the test set and ending with another vector in the test set and ideally the length of the sequence is a reasonably small number, at least in comparison with the length of the sequence used for random pattern testing (often of the order of 100,000 patterns or more)

This is the question which is addressed in the rest of this thesis In order to do more investigation, we give two more definitions here We know we have two kinds of circuits combinational circuits and sequential circuits Correspondingly, we also have two kinds of test sets combinational test sets and sequential test sets

Definition 3.5 A Combinational Test Set is a deterministic test set such that the order of applying the test vectors to the circuit to be tested does not affect the test result

Definition 3.6 A Sequential Test Set is a deterministic test set such that the order of applying the test vectors to the circuit to be tested does affect the test result

The complexity of finding a sequential test set and embedding it into a sequence from a LFSM is much higher, since the order of the vectors is important In the remainder of this thesis, we limit our attention to combinational test sets where the order of the vectors is irrelevant

Chapter 4

Investigations for Small Degree

LFSMs with primitive polynomials can generate m-sequences and m-sequences are very useful as test patterns. For convenience, we label such machines as primitive machines. Although there is no definitive proof that primitive LFSMs are better than other LFSMs as stimuli and compactors, this is concluded in recent research [15, 16]. In general, primitive LFSRs seem to behave much better than the non-primitive LFSRs both as stimuli and compactors.

From chapter 2 we know that LFSRs are the least expensive machines amongst the equivalence class of LFSMs. And, there are similarity transforms among LFSRs, LHCAs and general LFSMs. So, we can expand the behavior of LFSRs and LHCAs to general LFSMs. Then, we can conclude that, in general, primitive LFSMs behave better than non-primitive LFSMs both as stimuli and compactors. In the following chapters, we will only consider primitive LFSMs.

4.1 An Example

Suppose we are given the circuit shown in the Figure 4.1. One of the minimum test sets for

single stuck-at fault detection for this circuit is $T_1=\{0100, 0101, 0111, 1010, 1110\}$ From the previous chapter, we know that since this is a combinational test set, the order of the vectors does not matter

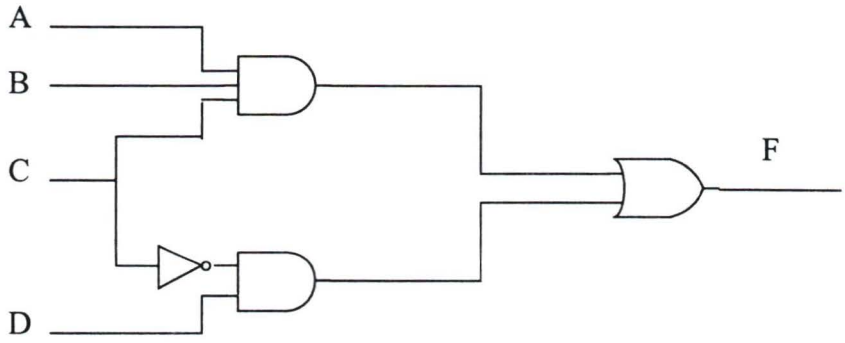

Figure 4 1 A 4-input circuit with F=ABC+$\overline{C}$D

The truth table for the circuit in Figure 4 1 is as shown in Table 4 1, and the highlighted rows are the test vectors in T_1

A	B	C	D	F
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

Table 4 1 The truth table for the circuit in Figure 4 1

We introduce the definition of *test embedding, minimal test embedding* and *t-length*

Definition 4.1 Suppose we have a test set $T = \{t_1, t_2, \ldots, t_n\}$ with n m-bit vectors. And we have a sequence $S = \{s_1, s_2, \ldots, s_N\}$ generated by a primitive LFSM, such that $N = 2^m - 1$. Any $S^* \subset S$ such that $t_i \in S^*$ for all i is a *test embedding* of T into S. The smallest such S^* is defined to be the *minimal test embedding* and its length is the *t-length* of T with respect to S.

Note that the test vectors in T do not necessarily occur in the same order in S. Our goal is to find the appropriate machine S which will give the smallest *t-length* of a given T over all possible choices of S. Consequently, this definition is only suitable for a combinational test set.

Consider our example function. Since it has four inputs, we need a 4-stage LFSM. Suppose we connect S_1 to A, S_2 to B, S_3 to C, and S_4 to D. For degree 4, we have two primitive polynomials, $x^4 + x^3 + 1$ and $x^4 + x + 1$. We have two types of LFSRs, and of LHCAs. Hence we can get four primitive LFSRs and four primitive LHCAs.

For all these primitive LFSRs, we can get their sequences (Table 4.2). For convenience, we use the corresponding decimal number to represent the state of each LFSR. The test set T_1 is $\{4, 5, 7, 10, 14\}$, and the highlighted cells are the test vectors in T_1. By inspecting Table 4.2, we see that we need 10 rows of column one to embed T_1, 11 rows for column two, 9 rows for column three, and 10 rows for column four. So by the definition of *t-length,* we find that *t-length* of T_1 with respect to S is 9. And sequence S is generated by the LFSR(I) with primitive polynomial $x^4 + x + 1$. The transition matrix for this LFSR is

$$T = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \end{bmatrix},$$

Time	LFSR(I) with x^4+x^3+1	LFSR(II) with x^4+x^3+1	LFSR(I) with x^4+x+1	LFSR(II) with x^4+x+1
t_1	1	1	1	1
t_2	9	8	12	8
t_3	13	12	6	4
t_4	15	14	3	2
t_5	14	15	13	9
t_6	7	7	10	12
t_7	10	11	5	6
t_8	5	5	14	11
t_9	11	10	7	5
t_{10}	12	13	15	10
t_{11}	6	6	11	13
t_{12}	3	3	9	14
t_{13}	8	9	8	15
t_{14}	4	4	4	7
t_{15}	2	2	2	3
t-length	10	11	9	10

Table 4.2 The t-lengths of T_1 with respect to sequences generated by 4-stage LFSRs

and its circuit diagram is shown in Figure 4.2

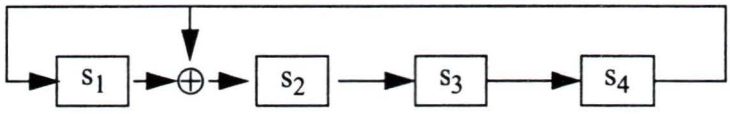

Figure 4.2 The diagram of LFSR(I) with primitive polynomial x^4+x+1

Similarly, we can get the corresponding sequences for these four primitive LHCAs

(Table 4 3) Again, the highlighted cells are the test vectors in test set T_1

Time	LHCA with 1010	LHCA with 0101	LHCA with 1101	LHCA with 1011
t_1	1	1	1	1
t_2	2	3	3	3
t_3	7	6	6	4
t_4	15	11	11	10
t_5	3	2	10	11
t_6	5	5	9	8
t_7	8	13	15	12
t_8	12	9	4	6
t_9	6	7	14	13
t_{10}	13	8	7	5
t_{11}	4	4	8	9
t_{12}	10	14	12	15
t_{13}	11	15	2	2
t_{14}	9	12	5	7
t_{15}	14	10	13	14
t-length	11	10	10	10

Table 4 3 The t-lengths of T_1 with respect to sequences generated by 4-stage LHCAs

From Table 4 3, by observation, we see that we need 11 rows of column one to embed the test set T_1, 10 rows for column two, three and four So, the primitive LHCAs with rules 0101, 1101, or 1011 can all generate a sequence S such that the *t-length* of T_1 is 10 with respect to it We arbitrarily choose the LHCA with rule 1101 as the generator of sequence S The transition matrix for this LHCA is

$$T = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix},$$

and its circuit diagram is shown in Figure 4.3

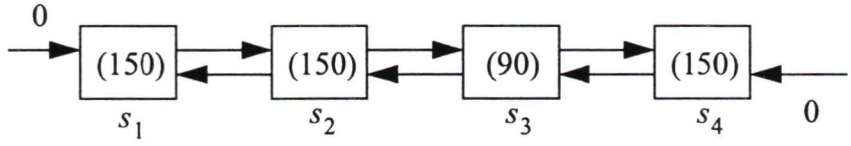

Figure 4.3 The diagram of LHCA with characteristic polynomial $x^4 + x^3 + 1$

However, we can find a primitive LFSM with primitive polynomial $x^4 + x^3 + 1$ which generates a sequence S such that *t-length* of T_1 is 6. The sequence S is shown in Table 4.4. The highlighted rows are the test vectors in T_1.

Time	S_1	S_2	S_3	S_4	Decimal
t_1	0	0	0	1	1
t_2	1	1	0	1	13
t_3	1	1	1	1	15
t_4	0	1	1	0	6
t_5	0	1	1	1	7
t_6	1	0	1	0	10
t_7	0	1	0	1	5
t_8	0	0	1	1	3
t_9	0	1	0	0	4
t_{10}	1	1	1	0	14
t_{11}	1	0	1	1	11
t_{12}	1	0	0	0	8
t_{13}	1	1	0	0	12
t_{14}	0	0	1	0	2
t_{15}	1	0	0	1	9

Table 4.4 Sequence generated by a primitive LFSM

Although this machine has shorter t-length than either the LFSRs or the LHCAs, it also

has a more complex structure Its transition matrix is

$$T = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 \end{bmatrix},$$

and its diagram is shown in Figure 4 4

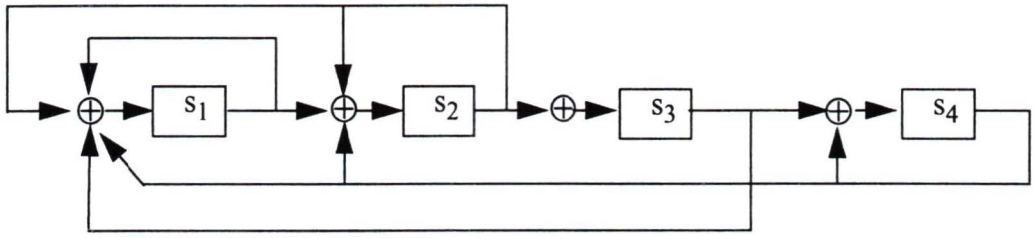

Figure 4 4 The diagram of LFSM with characteristic polynomial $x^4 + x^3 + 1$

From chapter 2, we know that LFSRs have the least expensive implementation of any machine in the particular class From the above example, we see that the LFSR(I) with characteristic polynomial $x^4 + x + 1$ gives t-length 9, the LHCAs with rules 0101, 1101, or 1011 give t-length 10, and a particular LFSM gives shorter t-length 6 So this particular LFSM is better than the LFSR and the LHCAs because it gives the shortest t-length for our example But on the other hand, comparing Figures 4 2, 4 3, and 4 4, we see that the particular LFSM has a much more complex structure than the LFSR and the LHCA This means when we reduce the t-length, we may increase the chip area We always need to balance the trade-off between the time and area

4.2 Phaseshifts between the Bitstreams

In the previous section, we give some sequences generated by primitive LFSMs. By comparing the bitstreams of each stage, we see that they all have the same bitstream but with different starting position. Is this true for all the sequences generated by the primitive LFSMs? In Table 4.5 we give the bitstreams for each stage of the 4-stage LHCA (which of course is an LFSM) with primitive polynomial $x^4 + x^3 + 1$.

Time	S_1	S_2	S_3	S_4	Decimal
t_1	0	0	0	1	1
t_2	0	0	1	1	3
t_3	0	1	1	0	6
t_4	1	0	1	1	11
t_5	1	0	1	0	10
t_6	1	0	0	1	9
t_7	1	1	1	1	15
t_8	0	1	0	0	4
t_9	1	1	1	0	14
t_{10}	0	1	1	1	7
t_{11}	1	0	0	0	8
t_{12}	1	1	0	0	12
t_{13}	0	0	1	0	2
t_{14}	0	1	0	1	5
t_{15}	1	1	0	1	13

Table 4.5 The bitstreams of the LHCA with $x^4 + x^3 + 1$.

From Table 4.5, we see that, starting with each highlighted cell, every stage has the same bitstream 000111101011001. The only difference is the starting position. In fact, previous work has shown that the bitstream from any stage of any LFSR or LHCA is identical, independent of the implementation [12], [13], [14]. The bitstream is only dependent

on the characteristic polynomial. If an LFSR and an LHCA have the same characteristic polynomial, they have the same bitstream [13], [14]. Paul H. Bardell discusses phaseshifts between the bitstreams in [19]. Since LFSRs, LHCAs and all other LFSMs are in an equivalence class and there are similarity transforms amongst them, it follows that for any n-stage primitive LFSM, the bitstream (it is the m-sequence since our machines are primitive) for each stage is identical except for the starting positions. Then, we can characterize the primitive LFSMs by the starting position of each m-sequence. We can arbitrarily choose the starting position of the m-sequence. From Table 4.5, if we arbitrarily choose the m-sequence as 000111101011001, then the starting position in column one for this m-sequence is 1, the starting position in column two is 4, the starting position in column three is 14, and the starting position in column four is 11. So, we can use the characteristic sequence (1, 4, 14, 11)-the index of time to represent our LFSM in Table 4.5. If we change the starting point of our m-sequence, the characteristic sequence is changed also. For example, if our m-sequence is 111101011001000, our characteristic sequence for our LFSM in Table 4.5 is (4, 7, 2, 14), and it represents the same machine as (1, 4, 14, 11) which using different m-sequence. So, every time we use the characteristic sequence to represent the LFSM, we need to include the m-sequence.

From Table 4.5, by observation, we see that we can use the vectors which are following the unit vectors to get the transition matrix of the LFSMs. In Table 4.5, the unit vectors are t_1 (0001), t_8 (0100), t_{11} (1000), and t_{13} (0010). Consider that the transition matrix for $x^4 + x^3 + 1$ which is

$$T = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix},$$

and in Table 4.5 the vectors following the unit vectors are t_{12} 1100, t_9 1110, t_{14} 0101, and t_2 0011. Combining those state vectors, we get

$$T' = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix},$$

and T and T' are the same This is a useful general result which is proved below

Theorem 4.1 If we have the maximum-length cycle of a primitive LFSM, we can get its transition matrix by using the next state vectors of the unit vectors

Proof Let $u_1 = (1, 0, \ldots, 0)$, $u_2 = (0, 1, \ldots, 0)$, $\ldots$, $u_m = (0, 0, \ldots, 1)$ be the unit vectors, $T = \begin{bmatrix} t_{11} & t_{12} & \ldots & t_{1m} \\ t_{21} & t_{22} & \ldots & t_{2m} \\ \ldots & \ldots & \ldots & \ldots \\ t_{m1} & t_{m2} & \ldots & t_{mm} \end{bmatrix}$ be the transition matrix, u_i^+ be the next vector of the unit vector u_i, and $u_i^+ = u_i \; T$ Therefore since

$$u_1 \; T = (t_{11}, t_{12}, \ldots, t_{1m})$$
$$u_2 \; T = (t_{21}, t_{22}, \ldots, t_{2m})$$
$$\ldots$$
$$u_m \; T = (t_{m1}, t_{m2}, \ldots, t_{mm})$$

and

$$u_i \; T = u_i^+$$

So the first row of T is the state vector following u_1, the second row of T is the state vector following u_2, and similarly for the other rows

Q E D

From theorem 4.1, we can derive the transition matrix of a primitive LFSM by using its state vectors Also, the primitive LFSMs have identical m-sequences for each stage Then, we can "construct" the primitive LFSMs using the m-sequence we know and derive the transition matrices of them using their state vectors But how do we construct the machines and how do we know if they are primitive? Since we use the starting positions of

the m-sequences to represent the primitive LFSMs, we may get another primitive LFSM if we change one starting position in the characteristic sequence. This change is the same as shifting the bitstreams for some stages. So based on Table 4.5, if we shift the m-sequence up or down arbitrarily, can we get another primitive LFSM? For example, if we shift 5 bits down for stage S_2 in Table 4.5, we get the sequence in Table 4.6.

Time	S_1	S_2	S_3	S_4	Decimal
t_1	0	0	0	1	1
t_2	0	1	1	1	7
t_3	0	0	1	0	2
t_4	1	1	1	1	15
t_5	1	1	1	0	14
t_6	1	0	0	1	9
t_7	1	0	1	1	11
t_8	0	1	0	0	4
t_9	1	0	1	0	10
t_{10}	0	0	1	1	3
t_{11}	1	0	0	0	8
t_{12}	1	1	0	0	12
t_{13}	0	1	1	0	6
t_{14}	0	1	0	1	5
t_{15}	1	1	0	1	13

Table 4.6 The sequence generated by shifting S_2 5 bits down from Table 4.5

From Table 4.6, we see that it is a primitive LFSM since it still has the maximum-length cycle. If we use m-sequence 000111101011001, the characteristic sequence for this primitive LFSM is (1, 9, 14, 11). By using the state vectors following the unit vectors, we have its transition matrix $T = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix}$. The diagram for this primitive LFSM is

the m-sequences to represent the primitive LFSMs, we may get another primitive LFSM if we change one starting position in the characteristic sequence This change is the same as shifting the bitstreams for some stages So based on Table 4 5, if we shift the m-sequence up or down arbitrarily, can we get another primitive LFSM? For example, if we shift 5 bits down for stage S_2 in Table 4 5, we get the sequence in Table 4 6

Time	S_1	S_2	S_3	S_4	Decimal
t_1	0	0	0	1	1
t_2	0	1	1	1	7
t_3	0	0	1	0	2
t_4	1	1	1	1	15
t_5	1	1	1	0	14
t_6	1	0	0	1	9
t_7	1	0	1	1	11
t_8	0	1	0	0	4
t_9	1	0	1	0	10
t_{10}	0	0	1	1	3
t_{11}	1	0	0	0	8
t_{12}	1	1	0	0	12
t_{13}	0	1	1	0	6
t_{14}	0	1	0	1	5
t_{15}	1	1	0	1	13

Table 4 6 The sequence generated by shifting S_2 5 bits down from Table 4 5

From Table 4 6, we see that it is a primitive LFSM since it still has the maximum-length cycle If we use m-sequence 000111101011001 , the characteristic sequence for this primitive LFSM is (1, 9, 14, 11) By using the state vectors following the unit vectors, we have its transition matrix $T = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix}$ The diagram for this primitive LFSM is

shown in Figure 4 5

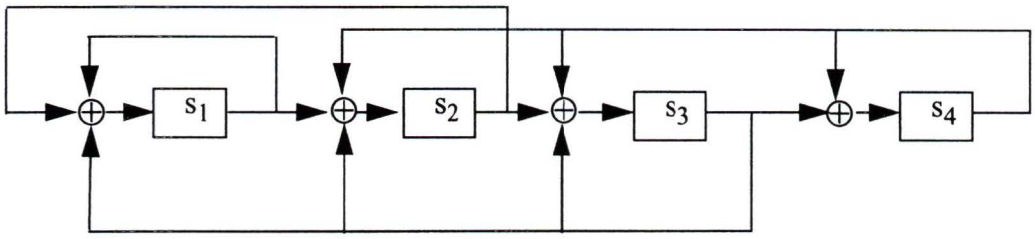

Figure 4 5 The diagram of LFSM with characteristic polynomial $x^4 + x^3 + 1$

However, if we shift 3 bits up for stage S_2, we get the sequence in Table 4 7

Time	S_1	S_2	S_3	S_4	Decimal
t_1	0	0	0	1	1
t_2	0	0	1	1	3
t_3	0	0	1	0	2
t_4	1	1	1	1	15
t_5	1	1	1	0	14
t_6	1	1	0	1	13
t_7	1	1	1	1	15
t_8	0	0	0	0	0
t_9	1	1	1	0	14
t_{10}	0	0	1	1	3
t_{11}	1	1	0	0	12
t_{12}	1	1	0	0	12
t_{13}	0	0	1	0	2
t_{14}	0	0	0	1	1
t_{15}	1	1	0	1	13

Table 4 7 The sequence generated by shifting S_2 3 bits up from Table 4 5

From Table 4 7, we see that row 8 is the all-0 vector, row 7 is a duplicate vector of row 4, row 12 is a duplicate vector of row 11, and so on We have duplicate vectors 1, 2, 3, 12,

13, 14, and 15. Clearly, this machine does not have a maximum-length cycle. Therefore, it is not a primitive LFSM. From this example, we see that if we use m-sequences to construct the LFSMs, the resulting machine is not primitive if

1. It does not have the maximum-length cycle.
2. It has all-0 vectors.
3. It has duplicated vectors.

Any of these three conditions is sufficient for us to judge that the LFSM is not primitive. However, we can establish some of these results more formally by considering state-space column rotations [7].

State-space column rotations

Let M be a primitive LFSM with state space S (S is $2^n - 1$ by n). Let R be a column rotation of S. The term vector always refers to a row of the state space or the rotated state space. Define R as good if there exists an LFSM M' such that the state space of M' is R. If R is good, the LFSM M' is similar to M. Denote the similarity transform by Q, so that SQ = R.

Lemma 4.2 R is good iff rank(R) = n.

Proof. First note that R is good iff there exists a similarity transform Q between S and R. If R is good, the rank of S (which is n) is preserved by Q, and so rank(R) = n.

To prove the converse, we show that the number of similarity transforms equals the number of rotations with rank n. Since no two similarity transforms give the same rotation, we conclude that good rotations are in 1 to 1 correspondence with rank n rotations.

The number of similarity transforms is exactly the number of invertible matrices, which is given by $N = \left(2^n - 1\right)\left(2^n - 2\right)\left(2^n - 4\right) \cdots \left(2^n - 2^{n-1}\right) = \frac{\left(2^n - 1\right)!}{\left(2^n - 2^{n-1} - 1\right)!}$.

A rotation has rank n iff the columns are linearly independent. We consider how the columns can be rotated so as to be linearly independent. Note that any column has $2^n - 1$ rotations. Supposing that k-1 columns have already been placed, the kth column can be

placed with any rotation that is not a linear combination of the previous k-1 columns Since every non-zero linear combination of the k-1 columns excludes a rotation of the kth column, and there are $2^{k-1}-1$ non-zero linear combinations, there are $\left(2^n-1\right)-\left(2^{k-1}-1\right)$ possible rotations This gives the total number of linearly independent rotations as $\left(2^n-1\right)\left(2^n-2\right)\left(2^n-4\right)\ \left(2^n-2^{n-1}\right)$, which equals N

Q E D

Theorem 4 3 R is good iff R contains all the unit vectors

Proof (⇒) Suppose R is good Since Q has full rank, Q defines a 1 to 1 correspondence between the non-zero state space and itself Hence R contains all non-zero vectors, including the unit vectors

(⇐) If R contains all n units vectors, then clearly R has rank n By Lemma 4 2, R is good

Q E D

Theorem 4 4 R is good iff R does not contain the zero vector

Proof (⇒) Suppose R is good Then Q has full rank, and cannot map a non-zero vector to the zero vector Hence SQ = R does not contain the zero vector

(⇐) We prove the contrapositive Suppose R is not good By lemma 4 2, R has rank k, with $1 \le k < n$ Without loss of generality, arrange the columns of R so that the first k columns are linearly independent

We claim that the first k columns of R contains a zero k-tuple To see this, consider any good rotation R' having the same first k columns as R (such a rotation is easily obtained) As R' is good, it contains the unit vector u_n, which is zero in its first k components Hence R also contains a zero k-tuple in the first k columns

Considering R again, the linear dependence of the last $n-k$ columns on the first k columns ensures that the remainder of the partially-zero vector is also zero

Q E D

According to theorem 4 4, if we have a sequence S generated by a primitive LFSM,

the m-sequence is dependent only on the characteristic polynomial, and since we can derive all the primitive polynomials, we should be able to get all the primitive LFSMs This is discussed in the section below

Time	S_1	S_2	S_3	S_4	Decimal
t_1	0	0	1	1	3
t_2	0	0	1	1	3
t_3	0	1	1	0	6
t_4	1	0	1	1	11
t_5	1	0	0	0	8
t_6	1	0	1	1	11
t_7	1	1	0	1	13
t_8	0	1	1	0	6
t_9	1	1	1	0	14
t_{10}	0	1	0	1	5
t_{11}	1	0	0	0	8
t_{12}	1	1	1	0	13
t_{13}	0	0	0	0	0
t_{14}	0	1	0	1	5
t_{15}	1	1	0	1	13

Table 4 9 The sequences generated by shifting S_3 1 bits up from Table 4 5

4.3 The Number of Primitive LFSMs

From the previous section, it follows that we should be able to get all the primitive LFSMs One way is doing the column rotations on all the m-sequences and using theorem 4 4 to judge the results Figure 4 6 shows an algorithm which can calculate the number of primitive LFSMs using column rotations

Step 1 Get all the primitive polynomials for degree n.

Step 2 Find all the m-sequences using the primitive LFSR or LHCA.

Step 3 For each m-sequence, do the column rotations and check for each result if it is a primitive LFSM according to theorem 4.4.

Figure 4.6 Algorithm to calculate the number of primitive LFSMs for degree n.

And Figure 4.7 gives the pseudo-code for step 3.

```
Define  N degree
Define  M length of m-sequence
Begin
    From 1 to M
        From 1 to M
        ...
        Begin
            set flag true,
            From 1 to M     /* total N for loop */
            Begin
                do the column rotation,
                if all N bits are 0
                    set flag false,
                    break,
            End
            if flag is true increase count,
        End
End
```

Figure 4.7 Pseudo-code of algorithm in Figure 4.5.

For the pseudo-code in Figure 4.6, when we do the column rotation, we do not rotate the first column in order to ensure we generate only non-duplicate primitive LFSMs. We know that we can represent a primitive LFSM using the start position of the m-sequences, and we need to refer to the m-sequence. For example, we can use (1, 9, 14, 11) to represent a 4-stage primitive LFSM in Table 4.5 with m-sequence 000111101011001. Obviously, (2, 10, 15, 12) with m-sequence 001111010110010 is the same machine as (1, 9, 14, 11) with 000111101011001 because we change the start position of the m-sequence from 000111101011001 to 001111010110010. And the start positions of the m-sequences are arbitrary. Therefore, our program only counts distinct primitive LFSMs.

For an n-stage LFSM, the maximum number of states it can achieve is $2^n - 1$. That is, the corresponding decimal states vary from 1 to $2^n - 1$. It follows that the maximum possible number of distinct primitive LFSMs is $\left(2^n - 2\right)!$, if we just looked at placing all the entries in arbitrary positions.

Table 4.10 shows the number of possible primitive LFSMs, the actual number of primitive LFSMs calculated by our simulation program, and the ratio of these two values for degree n less than 6.

Degree	Actual Number	Possible Number	Percentage
2	2	2	1.00
3	48	720	6.67×10^{-2}
4	2688	8.72×10^{10}	3.08×10^{-8}
5	1935360	2.65×10^{32}	7.30×10^{-27}

Table 4.10 The number of primitive LFSMs for degree less than 6.

In Table 4.10, the number of possible primitive LFSMs increases quickly when the degree n increases. At same time, the actual number of primitive LFSMs increases much more slowly than the possible number. We can see this from the percentage column. When

the degree equals 5, the ratio of actual number to the possible number is 7.30×10^{-27}. It follows that if we are given the numbers from 1 to $2^n - 1$, and we place them in an arbitrary order, the possibility of this resulting in a primitive LFSM is very small.

4.4 Statistics on Primitive LFSMs

For small sizes of degree n, we can find all the primitive LFSMs. And it is easy for us to do the systematic simulation. If we have the same model for both small sizes and large sizes of degree n, we can expand the conclusions from the small size to the large size. Since we can find all the primitive LFSMs for small size of degree n, we want to see if we can find a small t-length for any test set with n-bit test vectors over all primitive LFSMs. We do the simulation for degree n equal to 4. We only consider the combinational test sets in order to simplify our problem. We approach the simulation in the following manner,

For any given test set with m vectors, we want to see if we can find a machine which generates a sequence S

1. with t-length equal to m,
2. with t-length = m+1 if we can not find a machine satisfying 1,
3. with t-length = m+2 if we can not find a machine satisfying both 1 and 2.

Note that we only do the simulation on t-length up to m+2. This is because of limitations to our computing ability. When we do the simulation, we need to determine the value of m. If we choose m < 4, the simulation results will lose generality since m is too small. For m > 8, we can consider its complement set and (15 - m) < 8. Therefore, we choose m as 4, 5, 6, 7, and 8. Note that the degree n is 4 for the simulation. So, all the vectors in the test sets are 4-bit vectors. Then, it follows that we have $C_{15}^{m} = \frac{15!}{m!\,(15-m)!}$ possible combinational test sets for each m. Table 4.11 gives the total number of combinational test sets for each m.

Test Set Size m	Total Number of Combinational Test Sets
4	1365
5	3003
6	5005
7	6435
8	6435
Total	22243

Table 4.11 The total number of combinational test sets for different test set size for degree 4

For degree n equal to 4, we know from the previous section that there are 2688 primitive LFSMs. According to Table 4.11, we do the simulation on 22243 test sets. Then, for each of these test sets, is it possible to find a proper machine amongst the 2688 primitive LFSMs, such that this machine gives the t-length which equals its size m or m+1 or m+2? In order to answer this question, we use the algorithm in Figure 4.8. In this algorithm, we use the same method as in the previous section to find all the primitive LFSMs.

---

1. Initialize the counter of each test set as 0.

2. For each test set, if it appears consecutively in these primitive LFSMs, increase its counter. Calculate the total number of non-zero counters for all sets.

3. For those test sets whose counter is 0 after step 2, repeat step 2 by adding one vector to each of these test sets.

4. For those test sets whose counter is 0 after step 3, repeat step 2 by adding two vectors to each of these test sets.

Figure 4.8 Algorithm to calculate how many test sets have t-length equal to m, m + 1, or m +2 over all 2688 4-stage primitive LFSMs

Test Set Size	Total Number of Combinational Test Sets	Number of Test Sets Satisfy 1	Number of Test Sets Satisfy 2	Percentage	Number of Test Sets Satisfy 3	Percentage
4	1365	840	1260	0.92	1365	1.00
5	3003	1680	2835	0.94	3003	1.00
6	5005	2520	4480	0.90	4992	0.997
7	6435	2520	5070	0.79	N/A	N/A
8	6435	2520	N/A	N/A	N/A	N/A

Table 4.12 The simulation results for the algorithm in Figure 4.7

Table 4.12 shows the simulation results. Column one is the size of test set and it varies from 4 to 8. Column two is the total number of possible combinational test sets for each m. Note that by using equation $C_{15}^{m} = \frac{15!}{m!\,(15-m)!}$, when m equals 4 there are the same number of combinational test sets as when m equals 11. Similarly for m equal to 5 and 10, etc. Column three is the number of test sets which satisfy 1. That is, for any of these test sets, we can find at least one machine amongst the 2688 primitive ones such that it gives t-length m which equals its size. Note that row 3, row 4 and row 5 have the identical value 2520. Column four is the number of test sets which satisfy 2. That is, for any of these test sets, we can find at least one machine amongst the 2688 ones such that it gives t-length m+1. Column five is the ratio of column four and two. The smallest number is 0.79. This means that by adding one more vector to the original test set, we can find machines amongst the 2688 machines such that they give t-length m+1 for at least 70% of the possible test sets. Then, column six and seven report the numbers for condition 3. This time, the smallest percentage is 0.997. So, almost 90% of total test sets satisfy condition 3. These are all shown in Figure 4.9. Note that we have some missing entries for m equal to 7 and 8. This is because of limitations on our computing power.

According to Figure 4.9, it follows that, for any test set, the upper bound of the difference between the t-length and the test set size should not be bigger than 5. We give another

algorithm to calculate the biggest t-length for all the test sets with different sizes For every test set, there exists a smallest t-length over all primitive LFSMs We can group these numbers according to their test set size The maximum t-length is the largest number in each group This algorithm is shown in Figure 4 10 And Table 4 13 shows the results

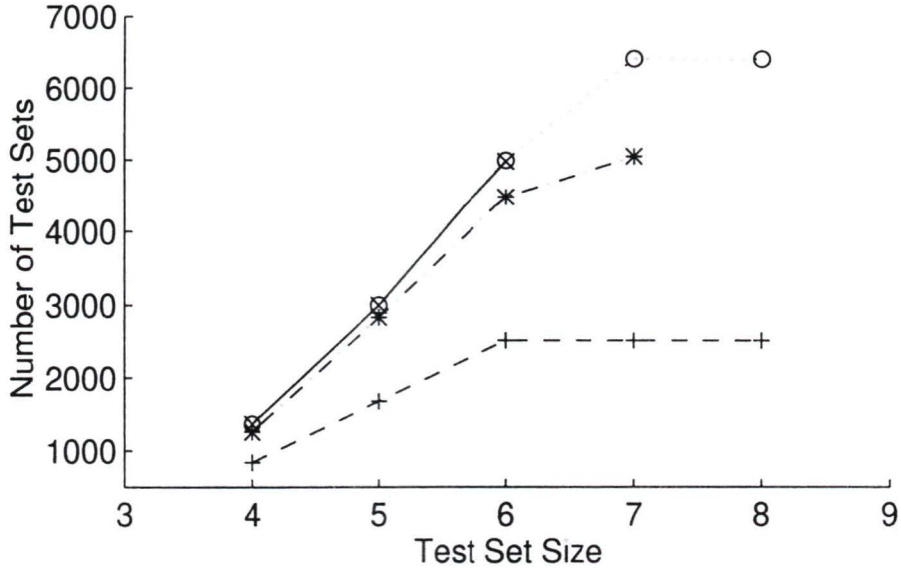

Figure 4 9 The simulation results,'---' for column 3 in Table 4 12, '- -' for column 4, and '····' for column 2, and '-'for column 6

---

1 Initialize count C for test sets of size m to be N

2 For each test set, find the maximum gap size G over all 2688 primitive LFSMs

3 If G < C, set C to be G

4 Repeat 2, 3 for all test sets of size m

5 The biggest t-length for all test sets of size m is N-C

6 Do 1-5 for all different size m

Figure 4 10 Algorithm to calculate the maximum t-length for all test sets with different size m

Size of the Test Set	Maximum T-Length	Difference
4	6	2
5	7	2
6	9	3
7	11	4
8	12	4
9	12	3
10	12	2
11	12	1

Table 4.13 The maximum t-lengths for all test sets with different size m

From Table 4.13, we see that, for all test sets with size m, the upper bound of the difference between the t-length and its size is 4. It follows that, for any combinational test sets with 4-bit test vectors, there exists at least one primitive LFSM such that it gives a t-length no bigger than its size plus 4. unfortunately, we do not know the proper way of finding these machines. According to theorem 4.1, one reasonable approach might be

1. If we have an all zero vector, omit that vector and only consider the rest of the vectors.
2. If we have all the unit vectors in the sequence, try to use the next vectors as the transition matrix.
3. If we do not have all the unit vectors in the sequence, add them into the sequence and repeat step 2.

Using this approach does not guarantee that it results in the best primitive LFSMs. However, if we know the distribution of all the t-lengths over all primitive LFSMs for all test sets, we also know the probability of choosing the proper machines if we randomly pick one. Figure 4.11 gives the distributions of all the t-lengths over all 2688 4-stage primitive LFSMs for all test sets with size 4, 5, and 7. Figure 4.12 gives the distributions of all the t-length over all possible primitive LFSMs, i.e. all the combinations of numbers from 1 to 15, for all test sets with size 4, 5, and 7.

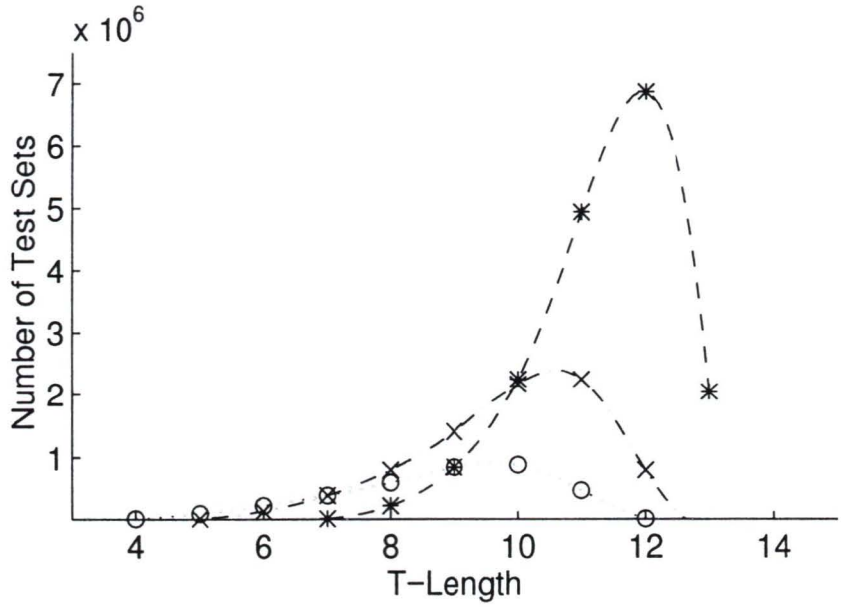

Figure 4 11 The distributions of the t-lengths of all test sets with size 4, 5 and 7 over 2688 4-Stage primitive LFSMs, '---' for size 7, '- -' for size 5, and '····' for size 4

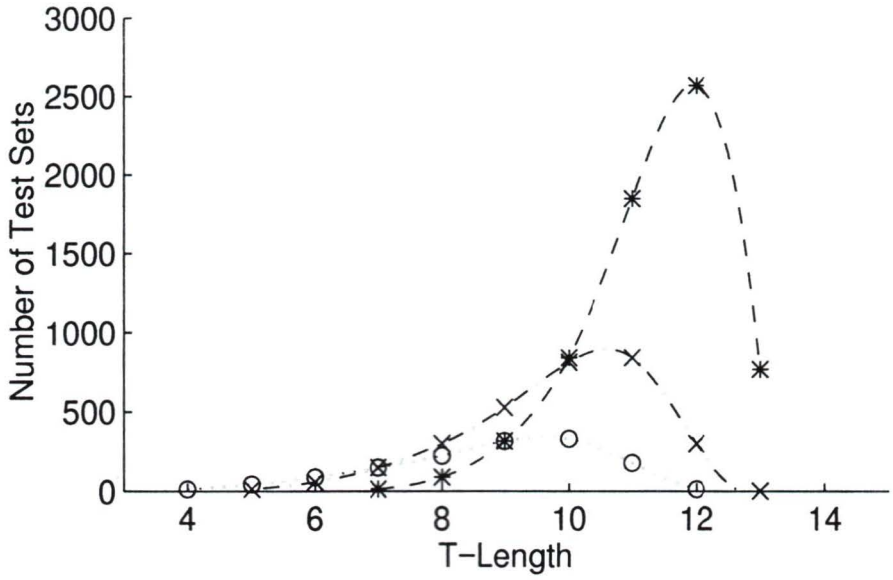

Figure 4 12 The distributions of the t-lengths of all test sets with size 4, 5 and 7 over all possible 4-stage primitive LFSMs,'---' for size 7, '- -' for size 5, and '····' for size 4

From Figure 4 11 and 4 12, we see the distributions in the two figures appears to be identical This is because the primitive LFSM generates the pseudo-random patterns We can see from the distribution curves that most test sets with size 4 have t-length 10, most test sets with size 5 have t-length 10 or 11, and most test sets with size 7 have t-length 12 Also, 85% of the test sets with size 4 have t-length less than 11, 89% of the test sets with size 5 have t-length less than 12, and 89% of the test sets with size 7 have t-length less than 13 Then, if we randomly pick a primitive LFSM for a test set, we will most likely get a t-length in the mid-range of the distribution curve

4.5 Summary

Generally, primitive LFSMs are better than the rest of the LFSMs as stimuli and compactors for testing digital circuits That is why we choose primitive LFSMs for our investigation In previous work, most research concerns the study of the formal properties of a particular type of LFSM Very little has been done in the opposite direction In this chapter, we investigate the relationship between the m-sequences and LFSMs We present some theorems about mapping a sequence to an LFSM

By doing the systematic investigation on degree n equal to 4, we can conclude that for any given 4-bit test set, there exists a primitive LFSM which gives a small t-length However, we do not have any effective method of finding this machine beyond a systematic search We present a reasonable approach And by providing the distributions of the t-length over 4-stage primitive LFSMs, we conclude that this approach is acceptable for small degree n

Chapter 5

Investigations for Large Degree

In the previous chapter, we investigate the problem for small degree. We give the relationship amongst the primitive polynomials, the m-sequences, and the number of primitive linear finite state machines. We present some theorems about the phaseshifts. We do the investigations for degree n equal to 4. According to the simulation results, we know that there exists a primitive LFSM which gives small t-length for any given 4-bit test set. However, the circuits on VLSI chips are often very complex. We may have more than 10 inputs. In this chapter, we first do some investigations for larger degree. Then, we present an alternative approach.

We first introduce two more complex combinational circuits and their test sets. One is the 74181 4-bit arithmetic logic unit with three different test sets, the other one is 8-bit ripple adder with its one test set. These are 14-input and 17-input circuits respectively for the two circuits. We simulate more than 7,500 sequences generated by LFSMs and analyze the results.

Then, we present an alternative way to approach our problem. We partition each test set into groups. For each group, we use a small degree LFSM. And we combine the state patterns together to get the original test set. This can reduce the t-length. However, we need to consider the added complexity of control which is required to get the correct com-

bination

5.1 Primitive Linear Finite State Machines

From the previous chapters, we know that we can get the total number of primitive polynomials by using equation $k(n) = \Phi\left(2^n - 1\right)/n$, where $\Phi(n) = n\sum_{p|n}(1 - 1/p)$ and n is the degree of the primitive polynomials. In chapter 4, we explore the relationship among the primitive polynomials, the m-sequences, and the number of primitive LFSMs. According to this relationship, we get an equation to calculate the number of primitive LFSMs for each degree n. In this chapter, we introduce the following theorem.

Theorem 5.1 Let n be the degree of polynomial, k(n) be the number of primitive polynomial for degree n, m(n) be the number of m-sequences for degree n, and f(n) be the number of primitive LFSMs for degree n. Then

$$k(n) = \Phi\left(2^n - 1\right)/n \text{ where } \Phi(n) = n\sum_{p|n}(1 - 1/p)$$

and

$$m(n) = k(n)$$

and

$$f(n) = (k(n))\left(2^n - 2\right)\left(2^n - 4\right)\left(2^n - 2^{n-1}\right) \text{ where n>1,}$$
$$f(1) = 1$$

Proof The first equation is proved in [6]. And from the definition of m-sequences and previous work of chapter 4, we know that the m-sequence is only dependent on the primitive polynomial. So the number of primitive polynomials determines the number of distinct m-sequences for each degree.

It is trivial for n=1. For n>1, from the previous chapters, we know that the number of primitive LFSMs equals the total number of distinct column rotations for all m-sequences. Since a rotation has rank n iff the columns are linearly independent, we consider how the columns can be rotated so as to be linearly independent. Note that any column has $2^n - 1$

rotations. Supposing that k-1 columns have been placed, the kth column can be placed with any rotation that is not a linear combination of the previous k-1 columns. Since every non-zero linear combination of the k-1 columns excludes a rotation of the kth column, and there are $2^{k-1}-1$ non-zero linear combinations, there are $\left(2^n-1\right)-\left(2^{k-1}-1\right)$ possible rotations. In order to get the distinct column rotations, we can not rotate the first column. So this gives the total number of primitive LFSMs for one primitive polynomial as $\left(2^n-2\right)\left(2^n-4\right)\ \left(2^n-2^{n-1}\right)$. Then, the total number of primitive LFSMs for degree n is the total number of primitive polynomials times the above number which gives $(k(n))\left(2^n-2\right)\left(2^n-4\right)\ \left(2^n-2^{n-1}\right)$.

Q E D

Table 5.1 shows the number of primitive polynomials, the number of m-sequences, and the number of primitive LFSMs for each degree n.

n	k(n)	m(n)	f(n)
1	1	1	1
2	1	1	2
3	2	2	48
4	2	2	2688
5	6	6	1935360
8	16	16	$\approx 3.36 \times 10^{17}$
14	756	756	$\approx 1.34 \times 10^{57}$
17	7710	7710	$\approx 1.69 \times 10^{85}$

Table 5.1 Total number of primitive polynomials, m-sequences and primitive LFSMs for different degrees.

From Table 5.1, we can see that the number of primitive LFSMs for each degree increases very quickly as the degree increases. For example, when n is 5, f(n) is 1,935,360. When n becomes 17, f(n) is 1.69×10^{85}. That is, when n increases by a factor of 3.40, f(n) increases by a factor of almost 10^{79}. This means that, for a given test set with large num-

bers of inputs, there exists a primitive LFSM which gives a small t-length However, if we randomly choose a primitive LFSM, the possibility of choosing the proper machine which gives the small t-length is vanishingly small

Note that we may have different connections between the inputs of a circuit under test and the stages of an LFSM Consider that if we have a 4-input combinational circuit, we use a 4-stage LFSM as the test pattern generator Then, we can connect our inputs for the circuit A, B, C, and D to the stages S_1, S_2, S_3, and S_4 of the LFSM We have the possibility of arbitrary connection pattern between them Consequently, we have 24 different connection patterns between the inputs and the stages If we use the column rotation idea, we can get all the possible connections with no extra work However, this may cause a more complex LFSM which requires extra XOR gates

5.2 Some Test Sets of 74181 and Simulation Results

5.2.1 74181 and Some Test Sets

The 74181 4-bit ALU (Figure 5 1 [17]) is one of the more complex standard combinational integrated circuits The circuit has 14 inputs, 8 outputs, 63 logic gates, 200 nodes (potential fault sites), and 400 single stuck-at faults [17] There are 10 distinct minimal complete single stuck-at fault test sets and we use three of them [18] These three deterministic test sets all have 14 14-bit vectors Table 5 2, Table 5 3 and Table 5 4 show these test sets

We want to find the smallest t-length for each test set over all primitive LFSMs Thus, for each test set, we need to get the corresponding t-length over every primitive LFSM, and find the smallest one amongst these numbers

5.2.2 Simulation Results

According to theorem 5 1, for degree n equal to 14, we have 756 primitive polynomials, and $1\ 34 \times 10^{57}$ primitive LFSMs Therefore, we have 1,512 primitive LHCAs, 756 prim-

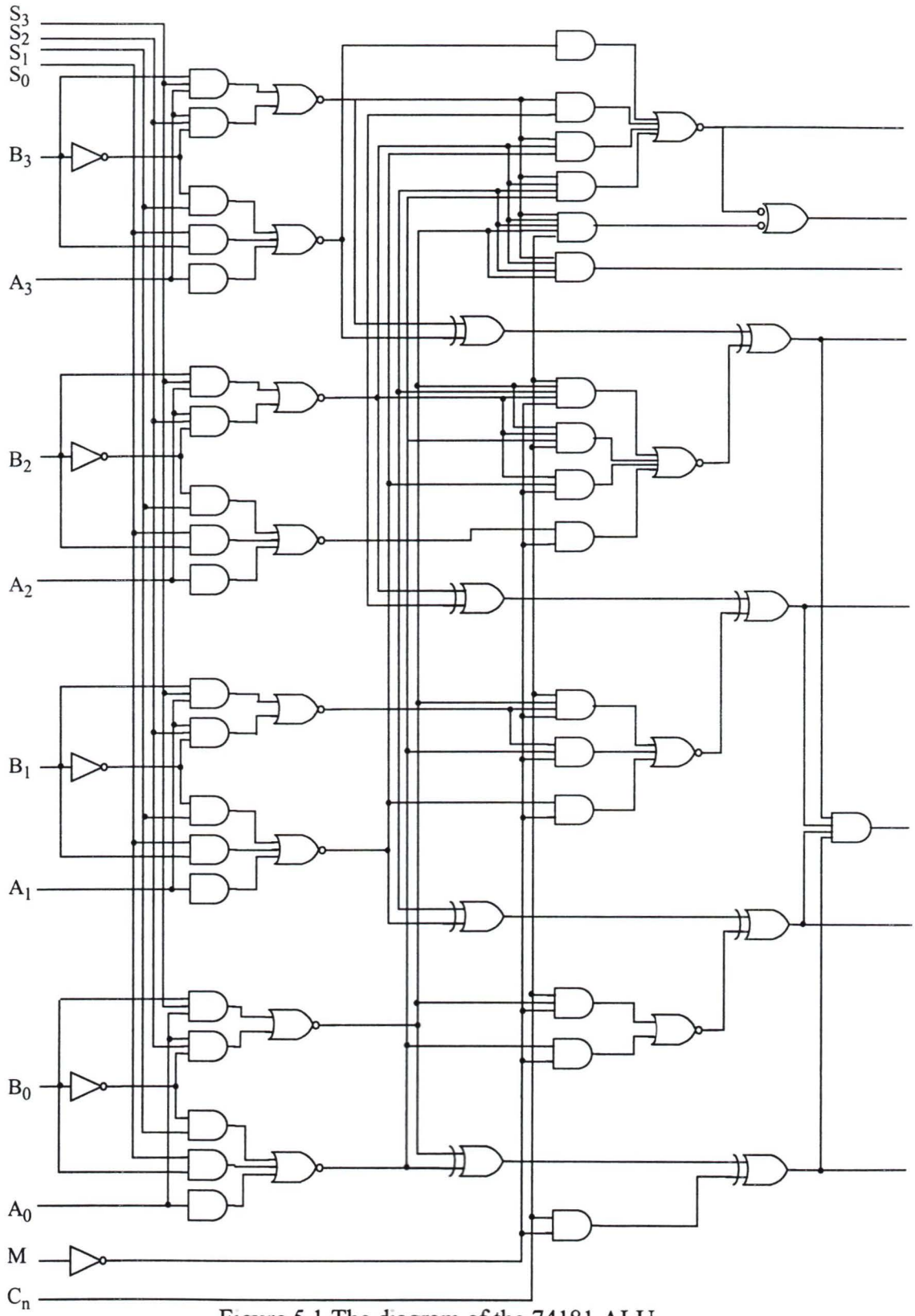

Figure 5 1 The diagram of the 74181 ALU

	M	S_0	S_1	S_2	S_3	C_n	A_0	B_0	A_1	B_1	A_2	B_2	A_3	B_3
1)	0	0	0	0	1	0	0	0	1	0	1	1	1	1
2)	0	0	1	1	0	1	0	0	0	0	0	0	0	1
3)	0	0	1	1	0	1	0	1	0	1	1	0	1	1
4)	0	0	1	1	0	1	1	0	1	1	1	1	0	0
5)	0	0	1	1	0	1	1	1	1	0	0	1	1	1
6)	0	1	0	0	1	0	0	0	1	0	0	1	1	1
7)	0	1	0	0	1	0	0	1	0	1	1	0	0	0
8)	0	1	0	0	1	0	1	0	0	0	1	0	0	1
9)	0	1	0	0	1	1	0	0	1	1	1	0	1	0
10)	1	0	0	0	1	0	1	0	1	0	1	0	1	0
11)	1	0	1	1	1	1	0	0	0	0	0	0	0	0
12)	1	0	1	1	1	1	0	1	0	1	0	0	1	0
13)	1	1	0	1	1	0	1	1	1	1	0	0	1	1
14)	1	1	0	1	1	1	0	0	0	1	0	1	0	1

Table 5.2: Test set 1 for the 74181 ALU

	M	S_0	S_1	S_2	S_3	C_n	A_0	B_0	A_1	B_1	A_2	B_2	A_3	B_3
1)	0	0	1	0	0	1	0	0	0	0	0	0	0	1
2)	0	0	1	0	0	1	0	0	0	0	0	0	1	1
3)	0	0	1	0	1	1	0	1	0	1	1	1	1	0
4)	0	0	1	0	1	1	1	0	1	1	0	1	0	0
5)	0	0	1	1	0	0	0	1	1	1	0	0	1	0
6)	0	1	0	0	1	1	1	1	1	0	1	0	0	1
7)	0	1	0	1	0	0	1	1	0	0	0	1	1	1
8)	0	1	0	1	0	1	0	0	1	0	1	1	1	1
9)	0	1	0	1	1	0	0	1	0	1	0	1	0	0
10)	1	0	0	1	0	0	1	1	1	1	1	1	1	1
11)	1	0	0	1	0	1	1	0	1	0	1	0	1	1
12)	1	0	1	1	1	1	0	1	0	1	0	0	1	1
13)	1	1	0	0	1	1	0	0	0	1	0	1	1	0
14)	1	1	0	1	1	0	1	0	0	0	0	0	1	1

Table 5.3: Test set 2 for the 74181 ALU

	M	S_0	S_1	S_2	S_3	C_n	A_0	B_0	A_1	B_1	A_2	B_2	A_3	B_3
1)	0	0	1	0	0	0	0	0	0	0	1	0	0	1
2)	0	0	1	0	1	0	0	0	0	0	0	0	0	0
3)	0	0	1	0	1	0	1	0	0	1	0	0	0	0
4)	0	0	1	0	1	1	0	1	1	1	1	0	1	0
5)	0	0	1	1	0	1	1	1	1	0	0	1	0	0
6)	0	1	0	0	1	0	0	0	1	0	0	1	1	1
7)	0	1	0	0	1	1	0	1	1	0	0	1	0	0
8)	0	1	0	1	0	1	0	0	0	0	1	0	1	1
9)	0	1	0	1	0	1	1	0	1	1	1	1	0	1
10)	1	0	1	0	0	1	1	0	1	0	1	0	1	1
11)	1	0	1	1	0	1	0	1	0	1	0	0	1	0
12)	1	1	0	0	1	1	0	0	0	1	1	0	0	1
13)	1	1	0	1	1	1	0	1	1	0	1	1	1	1
14)	1	1	0	1	1	1	1	1	1	0	0	0	1	0

Table 5.4 Test set 3 for the 74181 ALU

itive LFSR(I)s, and 756 primitive LFSR(II)s. It is not possible for us to simulate all the primitive LFSMs because of limitations on our computing power. But, we can easily simulate all the primitive LHCAs and primitive LFSRs because of the relatively small number. Since LHCAs and LFSRs are members in the equivalent class of LFSMs, their behavior can be extended to all the other members. So, we do our investigations on all 14-stage primitive LHCAs, all 14-stage primitive LFSRs, and some randomly picked primitive LFSMs.

From Table 5.2, 5.3 and 5.4, we see that, for each test vector, we can connect input M to stage S_1, input S_0 to stage S_2, ..., input B_3 to stage S_{14}. In fact, we can arbitrarily connect the inputs to the stages. Therefore, we have total of $14!$ ways of connecting inputs to stages of the machine. We can think of these connections as some new test sets derived from the original one. However, we can not do our simulations on all these $14!$ test sets because of the computational complexity. Consequently, we choose some of the test sets.

The simulation program includes three parts. First, we use Maple to generate all the

primitive polynomials for degree 14. Then, we use these primitive polynomials to generate the state patterns for the primitive LHCA, the primitive LFSR(I), and the primitive LFSR(II) for each primitive polynomial. Finally, we calculate the smallest t-length for each test set over all machines. Also, we generate 100 primitive LFSMs by using the algorithm in Figure 4.5 for each test set. We randomly choose 18 connection patterns from test set one, 17 connection patterns from test set two, and 18 connection patterns from test set three. Table 5.5 shows the smallest t-length for each connection pattern or test set over all primitive LHCAs, all primitive LFSR(I)s, all primitive LFSR(II)s, and 100 primitive LFSMs.

In Table 5.5, the first column indicates which test set by using a pair (A,B). A comes from the original test set number which is from 1 to 3. B, which is from 1 to 18, is the label for the different connection pattern between the inputs and stages of the machine. (See the Appendix A for detail of the connections). From Table 5.5, for test set one, we get the smallest t-length is 4,722, the largest t-length is 11,411, for test set two, the smallest t-length is 5,557, the largest t-length is 10,922, for test set three, the smallest t-length is 5,980, the largest t-length is 10,660. However, the t-lengths are mainly distributed over 6,000 to 8,000. Comparing with test set size 14, those numbers may appear to be comparatively large. However, a typical pseudo random test might often include over 100,000 patterns. So, 5,000-10,000 is a noticeable improvement.

5.3 The Test Set of the 8-Bit Ripple Adder and Simulation Results

5.3.1 8-Bit Ripple Adder and The Test Set

The 8-bit ripple adder contains eight small adders, each of which simply adds two inputs plus the incoming carry (Figure 5.2). There are 17 inputs, 9 outputs and total of 292 single stuck-at faults for this circuit.

Table 5.6 shows the deterministic test set which was generated by the Synopsys test

compiler and this test set detects all 292 single stuck-at faults

We want to know the smallest t-length for this test set Therefore, we need to get the t-lengths over all primitive LFSMs and find the smallest one from amongst these

Set	LHCA	LFSR (I)	LFSR (II)	LFSM	Set	LHCA	LFSR (I)	LFSR (II)	LFSM
(1,1)	8120	8334	7823	9331	(1,2)	7793	8625	8015	9142
(1,3)	7031	8150	5542	9262	(1,4)	7658	8217	8416	9300
(1,5)	7595	6689	8662	11411	(1,6)	7634	6217	7975	9055
(1,7)	7802	6621	6904	8551	(1,8)	7181	6602	7645	8601
(1,9)	7444	8584	6633	9352	(1,10)	6632	8329	7887	9656
(1,11)	7683	8393	7021	10363	(1,12)	6350	8160	6815	9924
(1,13)	7404	8547	8219	8760	(1,14)	7716	8065	8148	9995
(1,15)	7147	8676	8451	9865	(1,16)	7147	8676	8451	9073
(1,17)	6644	7716	7195	9208	(1,18)	7846	4722	7614	9013
(2,1)	7222	6274	7389	8605	(2,2)	6597	7349	7453	10598
(2,3)	6635	7734	8131	8624	(2,4)	7601	7261	8150	8344
(2,5)	6889	7623	8597	7626	(2,6)	6395	7692	6397	9484
(2,7)	7765	7445	7612	10090	(2,8)	6813	8355	8584	10410
(2,9)	8116	7096	8629	9307	(2,10)	7634	7241	6620	10048
(2,11)	8214	8418	8865	10922	(2,12)	7945	6853	8233	9350
(2,13)	6970	7647	8464	9038	(2,14)	7373	7410	7398	10557
(2,15)	7592	8258	6078	9484	(2,16)	6757	8465	6157	9052
					(2,18)	5557	6353	6578	8734
(3,1)	5980	6385	7966	10198	(3,2)	6750	8584	7910	8766
(3,3)	7895	7015	8205	9775	(3,4)	7004	7214	6991	8302
(3,5)	7767	8779	6417	8792	(3,6)	6577	7036	8153	10660
(3,7)	6660	7102	7440	9297	(3,8)	7956	7163	8727	9944
(3,9)	7760	6200	8556	9399	(3,10)	7393	8135	8638	8996
(3,11)	6969	7850	6553	7698	(3,12)	7806	8448	8338	9550
(3,13)	6922	7931	7404	9884	(3,14)	7324	7861	7636	10083
(3,15)	7858	6782	7883	9433	(3,16)	6860	8059	7795	9239
(3,17)	7254	8107	7725	8492	(3,18)	7619	6435	6603	9876

Table 5 5 T-lengths for test sets of the 74181

	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_0	CI
1)	0	1	0	1	0	1	1	0	0	1	0	1	1	0	0	0	0
2)	0	0	0	1	0	0	1	1	0	1	0	1	0	1	1	1	0
3)	0	0	0	0	1	1	0	1	0	0	0	0	0	0	1	1	1
4)	1	1	1	0	0	1	0	0	1	0	0	0	1	0	0	0	1
5)	0	0	1	0	1	0	1	1	1	0	0	1	1	0	0	1	1
6)	0	0	1	1	1	0	1	0	0	1	1	1	1	1	0	1	1
7)	1	0	1	1	1	1	0	0	0	0	1	0	0	1	0	0	1
8)	1	0	0	0	0	1	0	1	1	1	1	1	1	1	1	0	0
9)	0	1	0	1	0	0	0	0	0	0	1	1	0	0	0	1	0
10)	0	0	0	1	0	0	1	0	0	0	0	0	1	0	1	0	0
11)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
12)	0	1	0	1	0	0	0	1	1	1	0	0	0	0	0	0	0
13)	1	1	1	0	1	0	1	0	1	0	1	0	1	0	0	0	1
14)	1	0	1	0	1	1	0	1	0	1	0	0	1	1	1	0	1
15)	0	1	0	0	1	1	1	0	1	0	0	1	0	0	0	0	0
16)	1	0	0	1	0	0	0	1	1	1	1	1	1	1	0	1	1
17)	1	1	0	1	0	0	1	0	0	1	1	0	0	1	0	0	1

Table 5.6 Test set for the 8-bit ripple adder

5.3.2 Simulation Results

According to theorem 5.1, we have 7,710 primitive polynomials, and 1.69×10^{85} primitive LFSMs for degree n equal to 17. Therefore, we have 15,420 primitive LHCAs, 7,710 primitive LFSR(I)s, and 7,710 primitive LFSR(II)s. It is impossible for us to simulate all machines because of the computing complexity. Since we can easily simulate the state patterns of the primitive LHCAs and primitive LFSRs, we do our investigations on 1,500 17-stage primitive LHCAs, 1,500 17-stage primitive LFSR(I)s, and 1,500 17-stage primitive LFSR(II)s.

We have 17 inputs for this circuit. Therefore, we can have a total of 17! ways of connecting the inputs to the stages. This is a very large number and we can not do the simulations on all the possible connections. We choose 21 connection patterns to do the simulations.

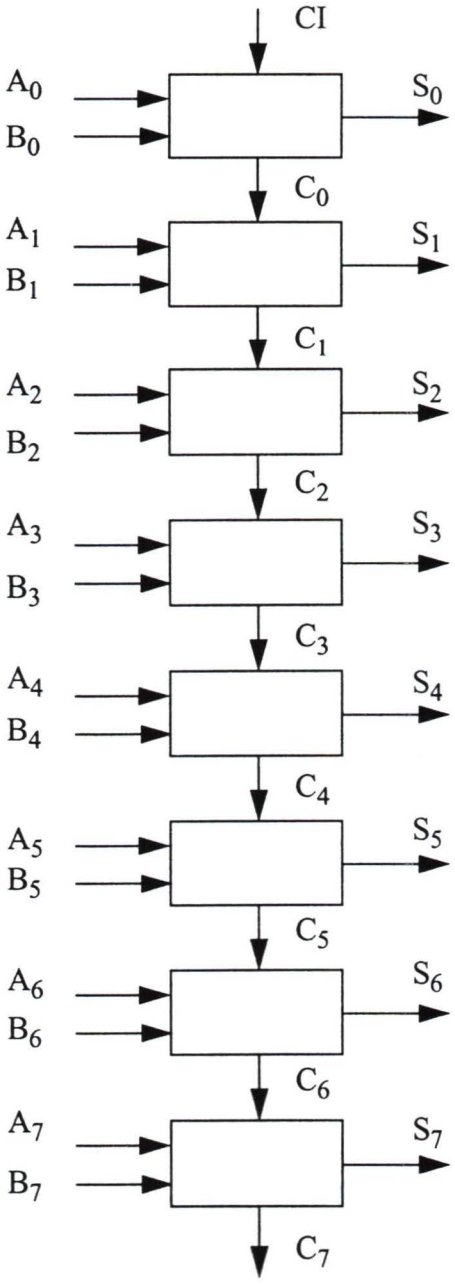

Figure 5 2 The diagram of the 8-bit ripple adder

The simulation program for this part is similar to the program in the 14-bit case However, the computational complexity is higher The simulation results are shown in Table 5 7

Set	LHCA	LFSR (I)	LFSR (II)	Set	LHCA	LFSR (I)	LFSR (II)
(1)	64203	60616	65682	(2)	71729	71937	70248
(3)	58447	65151	61153	(4)	65319	66966	71352
(5)	69694	52653	64699	(6)	69657	62048	66207
(7)	66639	60833	60316	(8)	60993	71150	64235
(9)	72042	68048	67882	(10)	72508	68379	68893
(11)	67519	64539	59173	(12)	67631	72639	65547
(13)	74514	70499	66402	(14)	62147	64854	66274
(15)	66729	60197	66185	(16)	71404	73533	67836
(17)	65208	69807	62390	(18)	68262	60616	66246
(19)	65738	66448	58812	(20)	61661	68508	65471
(21)	69761	53163	71153				

Table 5.7: T-lengths for the test set of the 8-bit ripple adder

In Table 5.7, the first column indicates the connections we choose for the simulation. We use numbers to label different test sets. (See Appendix B for detail of the connections). We get the smallest t-length is 52,653, and the largest t-length is 74,514. The t-lengths are mainly distributed over 60,000. Again, these numbers are comparatively large compared to 17. In the following section, we give the distribution of t-lengths for test sets with size 14.

5.4 Distribution of T-Lengths

In order to get the distribution of the t-lengths over all possible 14-stage LFSMs for any given 14-bit test set, we choose 2,000 sample test sets, run a simulation program using Maple, and get the distribution curve in Figure 5.3. From Figure 5.3, we see that the distribution curve is the same shape as the distribution curve for test sets with small size. In Figure 5.3, t-lengths occur mainly at 13,000 and are very rare for less than 7,500.

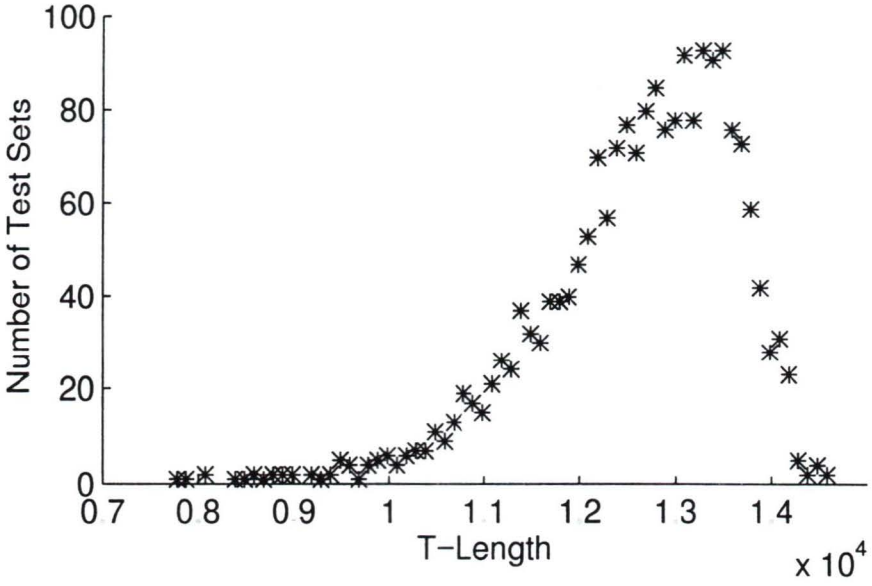

Figure 5 3 The distributions of t-lengths over all 14-stage possible LFSMs with 2000 sample connection Patterns

Since we know the shape of the distribution curves for t-lengths of any test sets, we can have a general idea about how large the t-length will be for any test set For example, if we are given a 14-bit test set, we know that the t-length for this set will be some number around 13,000 And the t-length is very unlikely to be smaller than 7,500

5.5 An Alternative Approach

From Chapter 4, if the size of the test vector is small, we can easily find a primitive LFSM which can give small t-length for any given test set This leads to an alternative approach for solving the problem for test sets with large size The idea here is partitioning We can partition large size test vectors into smaller size test vectors Then, we use several small stage machines to generate the small size vectors, and combine them together as our original test vectors

We can calculate the total sequence length by using this approach Suppose we originally have m test vectors each of which has n bits, we partition it into two parts with u bits

and v bits. Then, in the worst case, the total number is $2^{u-1} + 2^{v} \cdot (m-1)$. For our test sets of the 74181, if we partition one of its test sets into two parts with 7 bits and 7 bits, then, the worst number is $2^{6} + 2^{7} \cdot 13 = 1728$. Comparing this to the smallest t-length 4,722 in the previous section, the improvement is obvious.

5.5.1 Partitioning

The first step of this approach is partitioning. For our test set 3 of the 74181, we can partition it into two, or three, or four small test sets. However, the more small test sets we have, the more controls we need to consider. If the number of original inputs is smaller than 20, we can choose to partition the test set into two small ones. Therefore, we decide to partition test set 3 into two small test sets. Consequently, we partition it in the three most likely ways. Since the original test vectors have 14 bits, we can either partition it into 7 bit and 7 bit vectors, or 8 bit and 6 bit vectors, or 9 bit and 5 bit vectors. When we do the partitions, the increase of the maximum length of the sequence for one part will cause a decrease of the maximum length of the sequence for the other part.

5.5.2 Control the Machines

When we partition our test sets into smaller ones, we use several machines to generate the smaller ones, but we can not simply combine them together as the output pattern. Suppose we have smaller ones such as the following:

A	B
a_0	a_1
b_0	b_1
c_0	c_1
d_0	d_1
...	...

We use machine A to generate sequence A, and use machine B to generate sequence B.

We start machine A with a_0, and machine B with a_1. Then, we set machine A and B going at the same speed. After several cycles, we have b_0 for machine A. But most likely, we do not have b_1 for machine B at the same time. In order to get b_0b_1 together for the output, we need to hold machine A and let machine B continue to cycle until it reaches b_1. So, we need a way to control the machines.

Two obvious ways are possible. First, one only controls machine A. Every time when machine A reaches some vector i_0 in sequence A, hold machine A and let machine B cycle until it reaches i_1. Second, one controls both machines. Every time, hold the machine which reaches some vectors in sequence first and let the other machine cycle. In the next section, we gives the simulation results.

5.5.3 Simulation Results

We use test set 3 for the 74181 to do the simulation. We partition it into two parts with 7 bits/7 bits, 8 bits/6 bits, and 9 bits/5 bits. For each partition, we apply both control methods to control the machines. We do the simulation on all possible primitive LHCAs and primitive LFSRs and get the average number for the total sequence length. Table 5.8 shows the simulation results using control method one, and Table 5.9 shows the simulation results using control method two.

We start both machines at certain test vectors, and the choice of the test vectors can affect the total sequence length. In both tables, we use column one to indicate the partition method and the starting test vector in Table 5.4. For example, (7/7,1) indicates that we partition the original test set into two 7 bit parts and the starting vector is row one in Table 5.4. From Table 5.8, we see that the average number for the total sequence length for the 8 bits/6 bits partition is the shortest. In fact, during the simulation, we get the shortest sequence lengths as 412 for the 7/7 partition, 380 for the 8/6 partition, and 389 for the 9/5 partition. From Table 5.9, we see that the average number for the total sequence length for the 7/7 partition is the shortest. During the simulation, the shortest sequence length we get is 396 for the 7/7 partition.

We see that for the equal partition, (7/7 partition), using the control method two gets a shorter sequence. This is because we avoid some of the repeated cycles of machine B.

Set	LHCA	LFSR (I)	LFSR (II)	Set	LHCA	LFSR (I)	LFSR (II)
(7/7,1)	930 05	957 07	931 02	(7/7,2)	932 76	972 14	933 33
(7/7,3)	930 93	975 86	921 94	(7/7,4)	936 03	966 86	930 12
(7/7,5)	930 17	964 59	933 50	(7/7,6)	933 86	975 79	927 57
(7/7,7)	936 57	970 66	932 02	(7/7,8)	932 63	974 59	928 46
(7/7,9)	932 08	969 82	940 11	(7/7,10)	930 36	965 55	930 92
(7/7,11)	930 40	964 65	930 55	(7/7,12)	929 66	960 82	926 25
(7/7,13)	928 84	970 86	923 86	(7/7,14)	928 32	917 29	890 68
(8/6,1)	625 22	645 27	636 92	(8/6,2)	629 75	649 58	629 99
(8/6,3)	628 98	643 54	625 20	(8/6,4)	626 32	646 57	622 57
(8/6,5)	633 57	648 18	642 89	(8/6,6)	631 24	639 26	619 84
(8/6,7)	633 75	648 99	624 67	(8/6,8)	632 18	644 73	623 60
(8/6,9)	630 54	637 52	633 97	(8/6,10)	629 85	645 73	633 42
(8/6,11)	634 03	653 91	625 10	(8/6,12)	634 67	652 46	632 44
(8/6,13)	633 09	658 26	638 90	(8/6,14)	624 33	645 05	641 35
(9/5,1)	664 02	663 69	665 67	(9/5,2)	661 44	667 00	659 71
(9/5,3)	660 63	655 59	665 32	(9/5,4)	663 76	659 20	662 55
(9/5,5)	671 09	704 99	686 48	(9/5,6)	668 46	659 69	663 78
(9/5,7)	672 77	659 01	675 63	(9/5,8)	671 79	706 99	689 43
(9/5,9)	663 79	661 85	655 96	(9/5,10)	668 60	670 74	693 25
(9/5,11)	667 71	669 21	658 05	(9/5,12)	672 49	668 63	664 72
(9/5,13)	660 23	655 27	663 50	(9/5,14)	663 21	658 82	664 55

Table 5 8 Total sequence lengths for test set 3 of the 74181 using control method one

Set	LHCA	LFSR (I)	LFSR (II)	Set	LHCA	LFSR (I)	LFSR (II)
(7/7,1)	858 53	879 37	880 35	(7/7,2)	862 92	886 75	859 14
(7/7,3)	858 93	890 71	859 58	(7/7,4)	865 04	874 37	854 02
(7/7,5)	862 47	861 52	861 76	(7/7,6)	868 99	885 06	853 67
(7/7,7)	865 13	883 42	863 63	(7/7,8)	864 42	886 94	861 31
(7/7,9)	861 12	884 69	863 41	(7/7,10)	861 02	868 19	866 55
(7/7,11)	866 25	881 82	852 85	(7/7,12)	870 89	882 23	860 75
(7/7,13)	866 31	887 95	854 38	(7/7,14)	859 48	839 59	838 19
(8/6,1)	1439 35	1439 53	1402 41	(8/6,2)	1424 95	1430 69	1397 08
(8/6,3)	1414 47	1443 61	1414 78	(8/6,4)	1434 11	1453 02	1421 43
(8/6,5)	1430 19	1460 97	1437 82	(8/6,6)	1462 17	1413 61	1337 00
(8/6,7)	1411 29	1443 11	1362 55	(8/6,8)	1454 46	1431 35	1395 69
(8/6,9)	1435 39	1462 49	1428 89	(8/6,10)	1452 56	1423 29	1408 23
(8/6,11)	1444 98	1440 73	1398 90	(8/6,12)	1445 99	1384 15	1359 44
(8/6,13)	1437 85	1419 23	1505 93	(8/6,14)	1469 27	1454 75	1486 13
(9/5,1)	2992 28	2748 10	2895 05	(9/5,2)	2979 89	2795 47	2892 93
(9/5,3)	3008 75	2809 40	2929 50	(9/5,4)	3016 15	2892 04	2907 51
(9/5,5)	3008 71	2910 30	3020 84	(9/5,6)	2996 28	2875 65	2934 84
(9/5,7)	2966 77	2689 88	2904 72	(9/5,8)	3025 95	2868 37	2947 29
(9/5,9)	3011 71	2851 18	2856 79	(9/5,10)	3014 62	2911 25	2962 86
(9/5,11)	2988 68	2709 68	2847 65	(9/5,12)	2993 76	2838 76	2919 20
(9/5,13)	3034 28	2901 86	2934 77	(9/5,14)	3023 44	2710 45	2846 04

Table 5 9: Total sequence lengths for test set 3 of the 74181 using control method two

But for unequal partitions, (8/6 and 9/5 partition), using the control method one gets the shorter sequence. This is because method one does not cause the repeated cycles of

machine A which has a much longer sequence than machine B. Method two does cause a number of repeated cycles of machine A, since it is likely to encounter the required pattern for machine B first. However, we find that the average numbers of sequence lengths for the 7/7 partition with method two, the 8/6 and the 9/5 partitions with method one do not show much difference. Especially, the shortest sequence length for each group, 396, 380, and 389, is almost the same. Therefore, it appears we can use either an equal partition with control method two or a unequal partition with control method one to achieve our goal.

5.5.4 Complexity of Control

Using this alternative approach, we need to consider the complexity of the control that is required. If we partition the original test set into two groups, we need at least two controllers to control the machines for each part. Each controller controls the start time and the stop time for each machine in order to get the required state vectors. For example, we may get the following.

A	B
(0, 4)	(0,11)
(11,13)	(11,19)
(19,45)	(19,67)
...	...

and (m, n) indicates the start time and stop time for each machine. From the above, we start with a certain state vector. After 4 cycles, machine A encounters the first part of a required vector t. We then hold machine A and cycle machine B. After another 7 cycles, machine B reaches the other part of t. Therefore, we have vector t. Then, we restart machine A and allow B to continue and similarly proceed to find the rest of the vectors.

One of the implementations of the controller is to use AND gates and Flip-Flops and as is shown in Figure 5.4. From Figure 5.4, we see that we use some cells to store the value of the start time and the stop time. Then, we use these values as the inputs of an AND gate. The output of the AND gate sets a D Flip-Flop. Then, we use the outputs of the Flip-Flop to control the machine. Since it needs some memory storage, this implementa-

tion needs more area of the chip. There are a number of alternative control mechanisms, but they are outside the scope of our thesis.

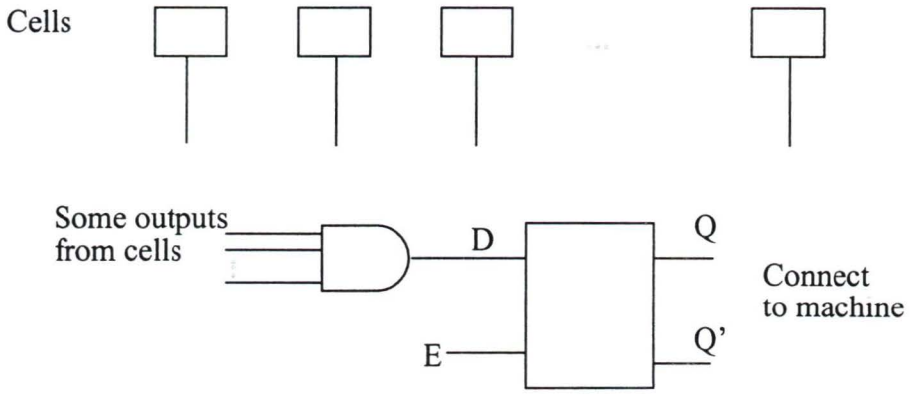

Figure 5.4 The diagram of one implementation of the controller.

5.6 Implementation

In the previous section, we present a reasonable approach which can reduce the total sequence length. In this section, we give the detail implementation of a real test set to show the alternative approach. We use the test set in Table 5.4. We partition it into two parts: one part has 8 bits, the other one has 6 bits. We choose columns M, S_0, S_1, S_2, C_n, A_0, B_0, A_1 as the first part. And we choose S_3, B_1, A_2, B_2, A_3, B_3 as the second part. We start both machines using the state vector 10 in Table 5.4. For convenience, we use the corresponding decimal number to represent the state vectors. Table 5.10 shows the partition results. In Table 5.10, the first column just labels the test vectors. Note it is not the same as column one in Table 5.4. We use an 8-stage primitive LHCA with rule 239 as machine A, and a 6-stage primitive LHCA with rule 22 as machine B.

We give part of the sequences of both machine in Table 5.11. All highlighted cells are the required test vectors. We use control method one and get the total sequence length of 380. Table 5.12 shows the start time and stop time for each vector. Note that the first column indicates the vectors corresponding to these in Table 5.10.

Test Vector	Machine A (8 bits)	Machine B (6 bits)	Test Vector	Machine A (8 bits)	Machine B (6 bits)
1)	174	11	8)	34	48
2)	181	18	9)	45	58
3)	196	57	10)	63	4
4)	221	47	11)	72	39
5)	223	34	12)	77	36
6)	32	9	13)	84	11
7)	32	32	14)	94	29

Table 5.10 The partition result for test set 3 of the 74181

Time	Machine A	Machine B	Time	Machine A	Machine B
0	174	11	14	208	45
1	165	17	15	8	8
2	189	58	16	28	20
3	137	59	17	58	54
4	223	57	18	67	33
5	30	62	19	228	18
6	61	53	20	94	63
7	73	36	21	221	55
8	255	30	22	25	35
9	110	37	23	55	21
10	133	28	24	82	52
11	205	34	25	207	38
12	49	23	26	54	25
13	91	51			

Table 5.11 Part of the sequences generated by machine A and B

	M	S_0	S_1	S_2	S_3	C_n	A_0	B_0	A_1	B_1	A_2	B_2	A_3	B_3
1)	0	0	1	0	0	0	0	0	0	0	1	0	0	1
2)	0	0	1	0	1	0	0	0	0	0	0	0	0	0
3)	0	0	1	0	1	0	1	0	0	1	0	0	0	0
4)	0	0	1	0	1	1	0	1	1	1	1	0	1	0
5)	0	0	1	1	0	1	1	1	1	0	0	1	0	0
6)	0	1	0	0	1	0	0	0	1	0	0	1	1	1
7)	0	1	0	0	1	1	0	1	1	0	0	1	0	0
8)	0	1	0	1	0	1	0	0	0	0	1	0	1	1
9)	0	1	0	1	0	1	1	0	1	1	1	1	0	1
10)	1	0	1	0	0	1	1	0	1	0	1	0	1	1
11)	1	0	1	1	0	1	0	1	0	1	0	0	1	0
12)	1	1	0	0	1	1	0	0	0	1	1	0	0	1
13)	1	1	0	1	1	1	0	1	1	0	1	1	1	1
14)	1	1	0	1	1	1	1	1	1	0	0	0	1	0

Table 5.4 Test set 3 for the 74181 ALU

itive LFSR(I)s, and 756 primitive LFSR(II)s. It is not possible for us to simulate all the primitive LFSMs because of limitations on our computing power. But, we can easily simulate all the primitive LHCAs and primitive LFSRs because of the relatively small number. Since LHCAs and LFSRs are members in the equivalent class of LFSMs, their behavior can be extended to all the other members. So, we do our investigations on all 14-stage primitive LHCAs, all 14-stage primitive LFSRs, and some randomly picked primitive LFSMs.

From Table 5.2, 5.3 and 5.4, we see that, for each test vector, we can connect input M to stage S_1, input S_0 to stage S_2, , input B_3 to stage S_{14}. In fact, we can arbitrarily connect the inputs to the stages. Therefore, we have total of $14!$ ways of connecting inputs to stages of the machine. We can think of these connections as some new test sets derived from the original one. However, we can not do our simulations on all these $14!$ test sets because of the computational complexity. Consequently, we choose some of the test sets.

The simulation program includes three parts. First, we use Maple to generate all the

primitive polynomials for degree 14. Then, we use these primitive polynomials to generate the state patterns for the primitive LHCA, the primitive LFSR(I), and the primitive LFSR(II) for each primitive polynomial. Finally, we calculate the smallest t-length for each test set over all machines. Also, we generate 100 primitive LFSMs by using the algorithm in Figure 4.5 for each test set. We randomly choose 18 connection patterns from test set one, 17 connection patterns from test set two, and 18 connection patterns from test set three. Table 5.5 shows the smallest t-length for each connection pattern or test set over all primitive LHCAs, all primitive LFSR(I)s, all primitive LFSR(II)s, and 100 primitive LFSMs.

In Table 5.5, the first column indicates which test set by using a pair (A,B). A comes from the original test set number which is from 1 to 3. B, which is from 1 to 18, is the label for the different connection pattern between the inputs and stages of the machine. (See the Appendix A for detail of the connections). From Table 5.5, for test set one, we get the smallest t-length is 4,722, the largest t-length is 11,411, for test set two, the smallest t-length is 5,557, the largest t-length is 10,922, for test set three, the smallest t-length is 5,980, the largest t-length is 10,660. However, the t-lengths are mainly distributed over 6,000 to 8,000. Comparing with test set size 14, those numbers may appear to be comparatively large. However, a typical pseudo random test might often include over 100,000 patterns. So, 5,000-10,000 is a noticeable improvement.

5.3 The Test Set of the 8-Bit Ripple Adder and Simulation Results

5.3.1 8-Bit Ripple Adder and The Test Set

The 8-bit ripple adder contains eight small adders, each of which simply adds two inputs plus the incoming carry (Figure 5.2). There are 17 inputs, 9 outputs and total of 292 single stuck-at faults for this circuit.

Table 5.6 shows the deterministic test set which was generated by the Synopsys test

compiler and this test set detects all 292 single stuck-at faults

We want to know the smallest t-length for this test set Therefore, we need to get the t-lengths over all primitive LFSMs and find the smallest one from amongst these

Set	LHCA	LFSR (I)	LFSR (II)	LFSM	Set	LHCA	LFSR (I)	LFSR (II)	LFSM
(1,1)	8120	8334	7823	9331	(1,2)	7793	8625	8015	9142
(1,3)	7031	8150	5542	9262	(1,4)	7658	8217	8416	9300
(1,5)	7595	6689	8662	11411	(1,6)	7634	6217	7975	9055
(1,7)	7802	6621	6904	8551	(1,8)	7181	6602	7645	8601
(1,9)	7444	8584	6633	9352	(1,10)	6632	8329	7887	9656
(1,11)	7683	8393	7021	10363	(1,12)	6350	8160	6815	9924
(1,13)	7404	8547	8219	8760	(1,14)	7716	8065	8148	9995
(1,15)	7147	8676	8451	9865	(1,16)	7147	8676	8451	9073
(1,17)	6644	7716	7195	9208	(1,18)	7846	4722	7614	9013
(2,1)	7222	6274	7389	8605	(2,2)	6597	7349	7453	10598
(2,3)	6635	7734	8131	8624	(2,4)	7601	7261	8150	8344
(2,5)	6889	7623	8597	7626	(2,6)	6395	7692	6397	9484
(2,7)	7765	7445	7612	10090	(2,8)	6813	8355	8584	10410
(2,9)	8116	7096	8629	9307	(2,10)	7634	7241	6620	10048
(2,11)	8214	8418	8865	10922	(2,12)	7945	6853	8233	9350
(2,13)	6970	7647	8464	9038	(2,14)	7373	7410	7398	10557
(2,15)	7592	8258	6078	9484	(2,16)	6757	8465	6157	9052
					(2,18)	5557	6353	6578	8734
(3,1)	5980	6385	7966	10198	(3,2)	6750	8584	7910	8766
(3,3)	7895	7015	8205	9775	(3,4)	7004	7214	6991	8302
(3,5)	7767	8779	6417	8792	(3,6)	6577	7036	8153	10660
(3,7)	6660	7102	7440	9297	(3,8)	7956	7163	8727	9944
(3,9)	7760	6200	8556	9399	(3,10)	7393	8135	8638	8996
(3,11)	6969	7850	6553	7698	(3,12)	7806	8448	8338	9550
(3,13)	6922	7931	7404	9884	(3,14)	7324	7861	7636	10083
(3,15)	7858	6782	7883	9433	(3,16)	6860	8059	7795	9239
(3,17)	7254	8107	7725	8492	(3,18)	7619	6435	6603	9876

Table 5 5 T-lengths for test sets of the 74181

	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_0	CI
1)	0	1	0	1	0	1	1	0	0	1	0	1	1	0	0	0	0
2)	0	0	0	1	0	0	1	1	0	1	0	1	0	1	1	1	0
3)	0	0	0	0	1	1	0	1	0	0	0	0	0	0	1	1	1
4)	1	1	1	0	0	1	0	0	1	0	0	0	1	0	0	0	1
5)	0	0	1	0	1	0	1	1	1	0	0	1	1	0	0	1	1
6)	0	0	1	1	1	0	1	0	0	1	1	1	1	1	0	1	1
7)	1	0	1	1	1	1	0	0	0	0	1	0	0	1	0	0	1
8)	1	0	0	0	0	1	0	1	1	1	1	1	1	1	1	0	0
9)	0	1	0	1	0	0	0	0	0	0	1	1	0	0	0	1	0
10)	0	0	0	1	0	0	1	0	0	0	0	0	1	0	1	0	0
11)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
12)	0	1	0	1	0	0	0	1	1	1	0	0	0	0	0	0	0
13)	1	1	1	0	1	0	1	0	1	0	1	0	1	0	0	0	1
14)	1	0	1	0	1	1	0	1	0	1	0	0	1	1	1	0	1
15)	0	1	0	0	1	1	1	0	1	0	0	1	0	0	0	0	0
16)	1	0	0	1	0	0	0	1	1	1	1	1	1	1	0	1	1
17)	1	1	0	1	0	0	1	0	0	1	1	0	0	1	0	0	1

Table 5.6 Test set for the 8-bit ripple adder.

5.3.2 Simulation Results

According to theorem 5.1, we have 7,710 primitive polynomials, and 1.69×10^{85} primitive LFSMs for degree n equal to 17. Therefore, we have 15,420 primitive LHCAs, 7,710 primitive LFSR(I)s, and 7,710 primitive LFSR(II)s. It is impossible for us to simulate all machines because of the computing complexity. Since we can easily simulate the state patterns of the primitive LHCAs and primitive LFSRs, we do our investigations on 1,500 17-stage primitive LHCAs, 1,500 17-stage primitive LFSR(I)s, and 1,500 17-stage primitive LFSR(II)s.

We have 17 inputs for this circuit. Therefore, we can have a total of 17! ways of connecting the inputs to the stages. This is a very large number and we can not do the simulations on all the possible connections. We choose 21 connection patterns to do the simulations.

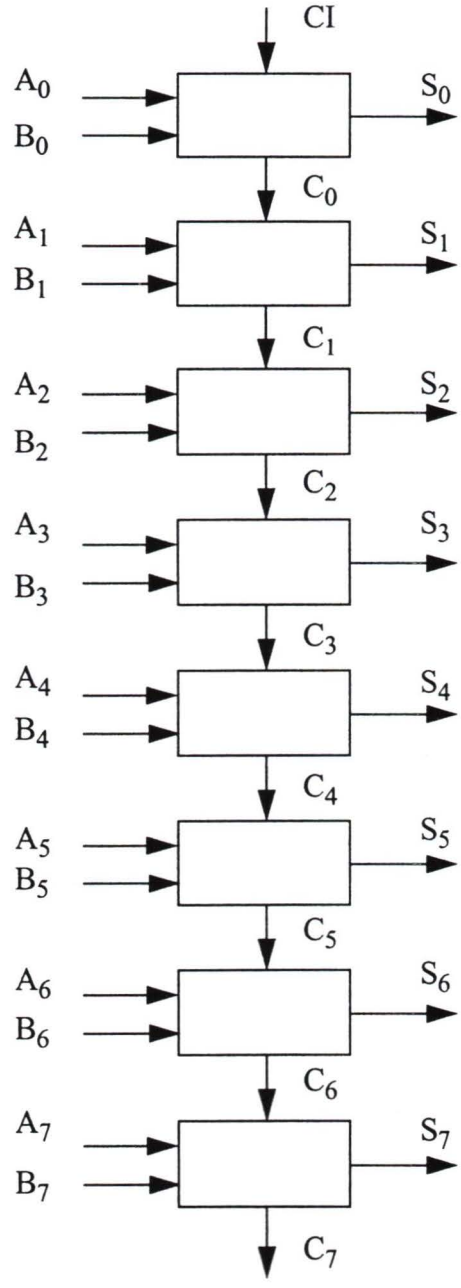

Figure 5.2 The diagram of the 8-bit ripple adder

The simulation program for this part is similar to the program in the 14-bit case. However, the computational complexity is higher. The simulation results are shown in Table 5.7.

Set	LHCA	LFSR (I)	LFSR (II)	Set	LHCA	LFSR (I)	LFSR (II)
(1)	64203	60616	65682	(2)	71729	71937	70248
(3)	58447	65151	61153	(4)	65319	66966	71352
(5)	69694	52653	64699	(6)	69657	62048	66207
(7)	66639	60833	60316	(8)	60993	71150	64235
(9)	72042	68048	67882	(10)	72508	68379	68893
(11)	67519	64539	59173	(12)	67631	72639	65547
(13)	74514	70499	66402	(14)	62147	64854	66274
(15)	66729	60197	66185	(16)	71404	73533	67836
(17)	65208	69807	62390	(18)	68262	60616	66246
(19)	65738	66448	58812	(20)	61661	68508	65471
(21)	69761	53163	71153				

Table 5.7: T-lengths for the test set of the 8-bit ripple adder

In Table 5.7, the first column indicates the connections we choose for the simulation. We use numbers to label different test sets. (See Appendix B for detail of the connections.) We get the smallest t-length is 52,653, and the largest t-length is 74,514. The t-lengths are mainly distributed over 60,000. Again, these numbers are comparatively large compared to 17. In the following section, we give the distribution of t-lengths for test sets with size 14.

5.4 Distribution of T-Lengths

In order to get the distribution of the t-lengths over all possible 14-stage LFSMs for any given 14-bit test set, we choose 2,000 sample test sets, run a simulation program using Maple, and get the distribution curve in Figure 5.3. From Figure 5.3, we see that the distribution curve is the same shape as the distribution curve for test sets with small size. In Figure 5.3, t-lengths occur mainly at 13,000 and are very rare for less than 7,500.

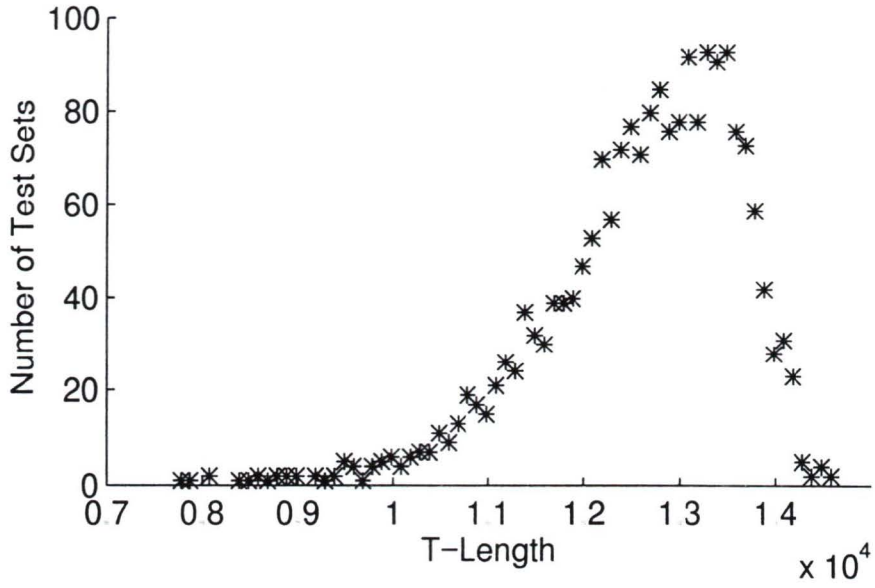

Figure 5.3 The distributions of t-lengths over all 14-stage possible LFSMs with 2000 sample connection Patterns.

Since we know the shape of the distribution curves for t-lengths of any test sets, we can have a general idea about how large the t-length will be for any test set. For example, if we are given a 14-bit test set, we know that the t-length for this set will be some number around 13,000. And the t-length is very unlikely to be smaller than 7,500.

5.5 An Alternative Approach

From Chapter 4, if the size of the test vector is small, we can easily find a primitive LFSM which can give small t-length for any given test set. This leads to an alternative approach for solving the problem for test sets with large size. The idea here is partitioning. We can partition large size test vectors into smaller size test vectors. Then, we use several small stage machines to generate the small size vectors, and combine them together as our original test vectors.

We can calculate the total sequence length by using this approach. Suppose we originally have m test vectors each of which has n bits, we partition it into two parts with u bits

and v bits. Then, in the worst case, the total number is $2^{u-1}+2^{v}\ (m-1)$. For our test sets of the 74181, if we partition one of its test sets into two parts with 7 bits and 7 bits, then, the worst number is $2^{6}+2^{7}\ 13 = 1728$. Comparing this to the smallest t-length 4,722 in the previous section, the improvement is obvious.

5.5.1 Partitioning

The first step of this approach is partitioning. For our test set 3 of the 74181, we can partition it into two, or three, or four small test sets. However, the more small test sets we have, the more controls we need to consider. If the number of original inputs is smaller than 20, we can choose to partition the test set into two small ones. Therefore, we decide to partition test set 3 into two small test sets. Consequently, we partition it in the three most likely ways. Since the original test vectors have 14 bits, we can either partition it into 7 bit and 7 bit vectors, or 8 bit and 6 bit vectors, or 9 bit and 5 bit vectors. When we do the partitions, the increase of the maximum length of the sequence for one part will cause a decrease of the maximum length of the sequence for the other part.

5.5.2 Control the Machines

When we partition our test sets into smaller ones, we use several machines to generate the smaller ones, but we can not simply combine them together as the output pattern. Suppose we have smaller ones such as the following.

A	B
a_0	a_1
b_0	b_1
c_0	c_1
d_0	d_1
...	...

We use machine A to generate sequence A, and use machine B to generate sequence B.

We start machine A with a_0, and machine B with a_1. Then, we set machine A and B going at the same speed. After several cycles, we have b_0 for machine A. But most likely, we do not have b_1 for machine B at the same time. In order to get b_0b_1 together for the output, we need to hold machine A and let machine B continue to cycle until it reaches b_1. So, we need a way to control the machines.

Two obvious ways are possible. First, one only controls machine A. Every time when machine A reaches some vector i_0 in sequence A, hold machine A and let machine B cycle until it reaches i_1. Second, one controls both machines. Every time, hold the machine which reaches some vectors in sequence first and let the other machine cycle. In the next section, we gives the simulation results.

5.5.3 Simulation Results

We use test set 3 for the 74181 to do the simulation. We partition it into two parts with 7 bits/7 bits, 8 bits/6 bits, and 9 bits/5 bits. For each partition, we apply both control methods to control the machines. We do the simulation on all possible primitive LHCAs and primitive LFSRs and get the average number for the total sequence length. Table 5.8 shows the simulation results using control method one, and Table 5.9 shows the simulation results using control method two.

We start both machines at certain test vectors, and the choice of the test vectors can affect the total sequence length. In both tables, we use column one to indicate the partition method and the starting test vector in Table 5.4. For example, (7/7,1) indicates that we partition the original test set into two 7 bit parts and the starting vector is row one in Table 5.4. From Table 5.8, we see that the average number for the total sequence length for the 8 bits/6 bits partition is the shortest. In fact, during the simulation, we get the shortest sequence lengths as 412 for the 7/7 partition, 380 for the 8/6 partition, and 389 for the 9/5 partition. From Table 5.9, we see that the average number for the total sequence length for the 7/7 partition is the shortest. During the simulation, the shortest sequence length we get is 396 for the 7/7 partition.

We see that for the equal partition, (7/7 partition), using the control method two gets a shorter sequence. This is because we avoid some of the repeated cycles of machine B.

Set	LHCA	LFSR (I)	LFSR (II)	Set	LHCA	LFSR (I)	LFSR (II)
(7/7,1)	930.05	957.07	931.02	(7/7,2)	932.76	972.14	933.33
(7/7,3)	930.93	975.86	921.94	(7/7,4)	936.03	966.86	930.12
(7/7,5)	930.17	964.59	933.50	(7/7,6)	933.86	975.79	927.57
(7/7,7)	936.57	970.66	932.02	(7/7,8)	932.63	974.59	928.46
(7/7,9)	932.08	969.82	940.11	(7/7,10)	930.36	965.55	930.92
(7/7,11)	930.40	964.65	930.55	(7/7,12)	929.66	960.82	926.25
(7/7,13)	928.84	970.86	923.86	(7/7,14)	928.32	917.29	890.68
(8/6,1)	625.22	645.27	636.92	(8/6,2)	629.75	649.58	629.99
(8/6,3)	628.98	643.54	625.20	(8/6,4)	626.32	646.57	622.57
(8/6,5)	633.57	648.18	642.89	(8/6,6)	631.24	639.26	619.84
(8/6,7)	633.75	648.99	624.67	(8/6,8)	632.18	644.73	623.60
(8/6,9)	630.54	637.52	633.97	(8/6,10)	629.85	645.73	633.42
(8/6,11)	634.03	653.91	625.10	(8/6,12)	634.67	652.46	632.44
(8/6,13)	633.09	658.26	638.90	(8/6,14)	624.33	645.05	641.35
(9/5,1)	664.02	663.69	665.67	(9/5,2)	661.44	667.00	659.71
(9/5,3)	660.63	655.59	665.32	(9/5,4)	663.76	659.20	662.55
(9/5,5)	671.09	704.99	686.48	(9/5,6)	668.46	659.69	663.78
(9/5,7)	672.77	659.01	675.63	(9/5,8)	671.79	706.99	689.43
(9/5,9)	663.79	661.85	655.96	(9/5,10)	668.60	670.74	693.25
(9/5,11)	667.71	669.21	658.05	(9/5,12)	672.49	668.63	664.72
(9/5,13)	660.23	655.27	663.50	(9/5,14)	663.21	658.82	664.55

Table 5.8 Total sequence lengths for test set 3 of the 74181 using control method one.

Set	LHCA	LFSR (I)	LFSR (II)	Set	LHCA	LFSR (I)	LFSR (II)
(7/7,1)	858 53	879 37	880 35	(7/7,2)	862 92	886 75	859 14
(7/7,3)	858 93	890 71	859 58	(7/7,4)	865 04	874 37	854 02
(7/7,5)	862 47	861 52	861 76	(7/7,6)	868 99	885 06	853 67
(7/7,7)	865 13	883 42	863 63	(7/7,8)	864 42	886 94	861 31
(7/7,9)	861 12	884 69	863 41	(7/7,10)	861 02	868 19	866 55
(7/7,11)	866 25	881 82	852 85	(7/7,12)	870 89	882 23	860 75
(7/7,13)	866 31	887 95	854 38	(7/7,14)	859 48	839 59	838 19
(8/6,1)	1439 35	1439 53	1402 41	(8/6,2)	1424 95	1430 69	1397 08
(8/6,3)	1414 47	1443 61	1414 78	(8/6,4)	1434 11	1453 02	1421 43
(8/6,5)	1430 19	1460 97	1437 82	(8/6,6)	1462 17	1413 61	1337 00
(8/6,7)	1411 29	1443 11	1362 55	(8/6,8)	1454 46	1431 35	1395 69
(8/6,9)	1435 39	1462 49	1428 89	(8/6,10)	1452 56	1423 29	1408 23
(8/6,11)	1444 98	1440 73	1398 90	(8/6,12)	1445 99	1384 15	1359 44
(8/6,13)	1437 85	1419 23	1505 93	(8/6,14)	1469 27	1454 75	1486 13
(9/5,1)	2992 28	2748 10	2895 05	(9/5,2)	2979 89	2795 47	2892 93
(9/5,3)	3008 75	2809 40	2929 50	(9/5,4)	3016 15	2892 04	2907 51
(9/5,5)	3008 71	2910 30	3020 84	(9/5,6)	2996 28	2875 65	2934 84
(9/5,7)	2966 77	2689 88	2904 72	(9/5,8)	3025 95	2868 37	2947 29
(9/5,9)	3011 71	2851 18	2856 79	(9/5,10)	3014 62	2911 25	2962 86
(9/5,11)	2988 68	2709 68	2847 65	(9/5,12)	2993 76	2838 76	2919 20
(9/5,13)	3034 28	2901 86	2934 77	(9/5,14)	3023 44	2710 45	2846 04

Table 5 9 Total sequence lengths for test set 3 of the 74181 using control method two

But for unequal partitions, (8/6 and 9/5 partition), using the control method one gets the shorter sequence This is because method one does not cause the repeated cycles of

machine A which has a much longer sequence than machine B. Method two does cause a number of repeated cycles of machine A, since it is likely to encounter the required pattern for machine B first. However, we find that the average numbers of sequence lengths for the 7/7 partition with method two, the 8/6 and the 9/5 partitions with method one do not show much difference. Especially, the shortest sequence length for each group, 396, 380, and 389, is almost the same. Therefore, it appears we can use either an equal partition with control method two or a unequal partition with control method one to achieve our goal.

5.5.4 Complexity of Control

Using this alternative approach, we need to consider the complexity of the control that is required. If we partition the original test set into two groups, we need at least two controllers to control the machines for each part. Each controller controls the start time and the stop time for each machine in order to get the required state vectors. For example, we may get the following:

A	B
(0, 4)	(0,11)
(11,13)	(11,19)
(19,45)	(19,67)
...	...

and (m, n) indicates the start time and stop time for each machine. From the above, we start with a certain state vector. After 4 cycles, machine A encounters the first part of a required vector t. We then hold machine A and cycle machine B. After another 7 cycles, machine B reaches the other part of t. Therefore, we have vector t. Then, we restart machine A and allow B to continue and similarly proceed to find the rest of the vectors.

One of the implementations of the controller is to use AND gates and Flip-Flops and as is shown in Figure 5.4. From Figure 5.4, we see that we use some cells to store the value of the start time and the stop time. Then, we use these values as the inputs of an AND gate. The output of the AND gate sets a D Flip-Flop. Then, we use the outputs of the Flip-Flop to control the machine. Since it needs some memory storage, this implementa-

tion needs more area of the chip There are a number of alternative control mechanisms, but they are outside the scope of our thesis

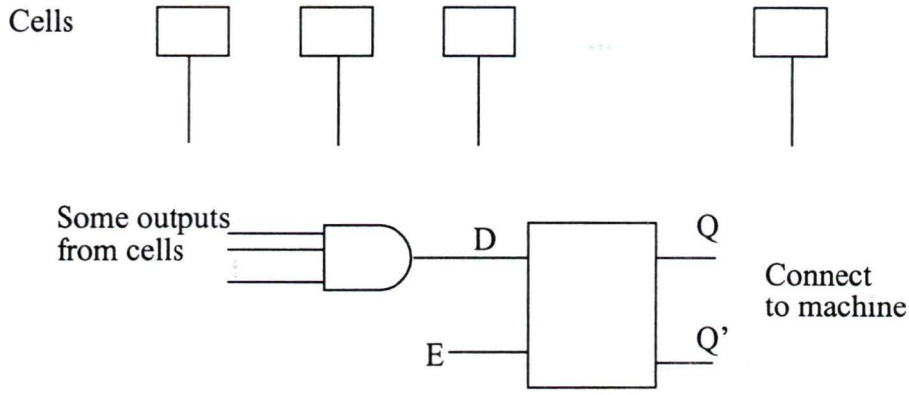

Figure 5 4 The diagram of one implementation of the controller

5.6 Implementation

In the previous section, we present a reasonable approach which can reduce the total sequence length In this section, we give the detail implementation of a real test set to show the alternative approach We use the test set in Table 5 4 We partition it into two parts one part has 8 bits, the other one has 6 bits We choose columns M, S_0, S_1, S_2, C_n, A_0, B_0, A_1 as the first part And we choose S_3, B_1, A_2, B_2, A_3, B_3 as the second part We start both machines using the state vector 10 in Table 5 4 For convenience, we use the corresponding decimal number to represent the state vectors Table 5 10 shows the partition results In Table 5 10, the first column just labels the test vectors Note it is not the same as column one in Table 5 4 We use an 8-stage primitive LHCA with rule 239 as machine A, and a 6-stage primitive LHCA with rule 22 as machine B

We give part of the sequences of both machine in Table 5 11 All highlighted cells are the required test vectors We use control method one and get the total sequence length of 380 Table 5 12 shows the start time and stop time for each vector Note that the first column indicates the vectors corresponding to these in Table 5 10

Test Vector	Machine A (8 bits)	Machine B (6 bits)	Test Vector	Machine A (8 bits)	Machine B (6 bits)
1)	174	11	8)	34	48
2)	181	18	9)	45	58
3)	196	57	10)	63	4
4)	221	47	11)	72	39
5)	223	34	12)	77	36
6)	32	9	13)	84	11
7)	32	32	14)	94	29

Table 5.10 The partition result for test set 3 of the 74181

Time	Machine A	Machine B	Time	Machine A	Machine B
0	174	11	14	208	45
1	165	17	15	8	8
2	189	58	16	28	20
3	137	59	17	58	54
4	223	57	18	67	33
5	30	62	19	228	18
6	61	53	20	94	63
7	73	36	21	221	55
8	255	30	22	25	35
9	110	37	23	55	21
10	133	28	24	82	52
11	205	34	25	207	38
12	49	23	26	54	25
13	91	51	...	...	...

Table 5.11 Part of the sequences generated by machine A and B

Chapter 6

Conclusion and Future Work

6.1 Conclusion

There is a considerable amount of ongoing research in the area of pseudorandom testing, but mainly on studying the behavior of LFSRs and LHCAs. However, little work has been done in investigating the relationship mapping the sequences to LFSMs. This is very important in the testing area because of the importance of the test generation. In this thesis, we investigate using a primitive LFSM to generate a sequence such that the test vectors for a circuit appear reasonably close together in this sequence.

We present some results and theorems related to mapping the sequences to LFSMs. We present the simulation results and their analysis. We also give an alternative approach to solving the problem. Based on the above, if we are given a deterministic test set, we have the following conclusions.

1. there must exist a primitive LFSM which can give a comparatively small t-length for this test set.
2. finding a proper machine such that it can give a small t-length for this test set is very difficult.
3. a randomly picked primitive LFSM often gives a comparatively long t-length.

Vector	Machine A	Machine B	Vector	Machine A	Machine B
1)	-	-	8)	(145, 151)	(145, 160)
5)	(0, 4)	(0, 11)	12)	(160, 191)	(160, 196)
14)	(11, 27)	(11, 38)	11)	(196, 222)	(196, 236)
4)	(38, 39)	(38, 44)	3)	(236, 247)	(236, 256)
13)	(44, 59)	(44, 63)	6)	(256, 281)	(256, 303)
10)	(63, 69)	(63, 99)	7)	(303, 303)	(303, 354)
2)	(99, 103)	(99, 145)	9)	(354, 362)	(354, 380)

Table 5.12 Start time and stop time for the controllers

We need a 9-input AND gate to set the Flip-Flop because the largest value in Table 5.12 is 380. The whole sequences generated by two machines is given in the Appendix C.

5.7 Summary

In this chapter, we present the equation for calculating the number of primitive linear finite state machine for each degree, give the simulation results for the t-lengths for the test sets with 14 14-bit vectors and 17 17-bit vectors, provide an alternative approach to solve the problem, and present an implementation using the alternative approach.

From the equation, we see that the number of primitive LFSMs is very large when the degree is bigger than 10. And from the simulation results of several real test sets, we get that simply using a primitive LFSM gives a comparatively long t-length for any test sets with more than 10 inputs.

Then, an alternative approach is presented. By giving the simulation results and analyzing the control complexity, we conclude that this alternative approach can reduce the total sequence length. However, it requires further investigation to determine if the extra area overhead required by the control circuitry justifies the decrease in the t-length.

4. The t-lengths are often shorter than the comparable pseudorandom sequence.

Although we do not provide a methodology for the generation of a deterministic test set using LFSMs, we do give some general conclusions and an alternative approach to this problem. The contribution of our research can be summarized as follows:

- It is the first study of the relationship between the sequences and LFSMs. We give some related results to derive primitive LFSMs by using m-sequences, getting the transition matrices of primitive LFSMs by using the state vectors, and we present an equation to calculate the number of primitive LFSMs for all degrees.
- Our research gives some direction for the solution of the problem of deriving a minimum embedding of a deterministic test set into a pseudorandom sequence generated by an LFSM. We point out the existence of such machines and the difficulty of finding them.
- We suggest that doing some modifications on LFSMs may solve the problem. We present an approach of partitioning the machines which reduces the length of the sequence.

6.2 Future Work

There are many interesting open questions remaining from our research. Some of them are:

- We present the distribution curve of the t-lengths over primitive LFSMs, which we are derived experimentally, in this thesis. However, a mathematical basis for predicting the t-length of a given test set on all possible primitive LFSMs would be valuable.
- It is known that, in general, primitive LFSMs are better than non-primitive LFSMs as the stimuli and compactors. Hence, we use primitive LFSMs in this thesis. However, non-primitive LFSMs may perform better in test set generation. An investigation using non-primitive LFSMs for test set generation would be a good research topic.
- We present an alternative approach using partitioning. More investigation needs to be done on partitioning the original test set into three, or more parts. An investigation of the required control structure is also required.

Bibliography

[1] H. S. Stone. *Discrete Mathematical Structures and Their Applications*. Science Research Associates, Inc., 1973.

[2] P. H. Bardell, W. H. McAnney, and J. Savir. *Built-in Test for VLSI Pseudorandom Techniques*. John Wiley and Sons, New York, 1987.

[3] M. Abramovici, M. A. Breuer, and A. D. Friedman. *Digital Systems Testing and Testable Design*. AT & T Bell Laboratories and W. H. Freeman and Company, 1990.

[4] S. Zhang, R. Byrne, J. C. Muzio, and D. M. Miller. "Why Cellular Automata Are Better Than LFSRs as Built-In Self-Test Generators for Sequential-Type Faults," *Proc. ISCAS'94*, May 1994.

[5] T. Slater, and M. Serra. "Tables of Linear Hybrid 90/150 Cellular Automata," *Technical report, DCS-105-IR, Department of Computer Science, University of Victoria*, Victoria, BC, Canada, 1990.

[6] S W Golomb *Shift Register Sequences* Holden-Day, Inc , 1967

[7] K Cattell Personal Communication, March, 1997

[8] M Serra, T Slater, J C Muzio, and D M Miller "The analysis of one dimensional linear cellular automata and their aliasing properties," *IEEE Transactions on Computer-Aided Design,* 9(7), July 1990

[9] D E Knuth *The Art of Computer Programming, Vol 2 Seminumerical Algorithms* Addison-Wesley, Reading, Mass, 1969

[10] C E Shannon *Communication Theory of Secrecy Systems* Bell Sys Tech J 28(4), 1949

[11] S W Golomb, et al *Digital Communications with Space Applications* Prentice-Hall, Englewood Cliffs, N J , 1967

[12] P H Bardell, "Design considerations for parallel pseudorandom pattern generators," *J Elect Testing Theory and Appl* , vol 1, pages 73-87, 1990

[13] M Serra, D M Miller, and J C Muzio, "Linear cellular automata and LFSRs are isomorphic," *Proc Third Tech Workshop on New Dir for IC Testing*, Halifax, N S pages 213-233, October, 1988

[14] P H Bardell "Analysis of cellular automata used as pseudorandom generators," In *Proc of the International Test Conference,* pages 762-768, 1990

[15] M Damiani, P Olivio, M Favalli, and B Ricco "An analytical model for the aliasing probability in signature analysis testing," *IEEE Transactions on Computers,*

8(11) 1133-1144, November 1989

[16] T W Williams, W Daehn, M Gruetznmer, and C W Starke "Comparison of aliasing errors for primitive and non-primitive polynomials," *International Test Conference,* pages 282-288, 1986

[17] E J McCluskey, "Tutorial on digital testing," *IEEE International Conference on Computer-Aided Design,* 1985

[18] R Bryant, Personal Communication, March 1997

[19] P H Bardell, "Discrete Logarithms A Parallel Pseudorandom Pattern Generator Analysis Method," *IBM Corporation,* Poughkeepsie, NY 12602, 1991

Appendix A

Connection Patterns Used in Section 5.2

Pattern One for All Three Test Sets

M	S_0	S_1	S_2	S_3	C_n	A_0	B_0	A_1	B_1	A_2	B_2	A_3	B_3
s_{14}	s_{13}	s_{12}	s_{11}	s_{10}	s_9	s_8	s_7	s_6	s_5	s_4	s_3	s_2	s_1

Pattern Two for All Three Test Sets

M	S_0	S_1	S_2	S_3	C_n	A_0	B_0	A_1	B_1	A_2	B_2	A_3	B_3
s_1	s_{14}	s_{13}	s_{12}	s_{11}	s_{10}	s_9	s_8	s_7	s_6	s_5	s_4	s_3	s_2

Pattern Three for All Three Test Sets

M	S_0	S_1	S_2	S_3	C_n	A_0	B_0	A_1	B_1	A_2	B_2	A_3	B_3
s_2	s_1	s_{14}	s_{13}	s_{12}	s_{11}	s_{10}	s_9	s_8	s_7	s_6	s_5	s_4	s_3

Pattern Four for All Three Test Sets

M	S_0	S_1	S_2	S_3	C_n	A_0	B_0	A_1	B_1	A_2	B_2	A_3	B_3
s_3	s_2	s_1	s_{14}	s_{13}	s_{12}	s_{11}	s_{10}	s_9	s_8	s_7	s_6	s_5	s_4

Pattern Five for All Three Test Sets

M	S_0	S_1	S_2	S_3	C_n	A_0	B_0	A_1	B_1	A_2	B_2	A_3	B_3
s_4	s_3	s_2	s_1	s_{14}	s_{13}	s_{12}	s_{11}	s_{10}	s_9	s_8	s_7	s_6	s_5

Pattern Six for All Three Test Sets

M	S_0	S_1	S_2	S_3	C_n	A_0	B_0	A_1	B_1	A_2	B_2	A_3	B_3
s_5	s_4	s_3	s_2	s_1	s_{14}	s_{13}	s_{12}	s_{11}	s_{10}	s_9	s_8	s_7	s_6

Pattern Seven for All Three Test Sets

M	S_0	S_1	S_2	S_3	C_n	A_0	B_0	A_1	B_1	A_2	B_2	A_3	B_3
s_6	s_5	s_4	s_3	s_2	s_1	s_{14}	s_{13}	s_{12}	s_{11}	s_{10}	s_9	s_8	s_7

Pattern Eight for All Three Test Sets

M	S_0	S_1	S_2	S_3	C_n	A_0	B_0	A_1	B_1	A_2	B_2	A_3	B_3
s_7	s_6	s_5	s_4	s_3	s_2	s_1	s_{14}	s_{13}	s_{12}	s_{11}	s_{10}	s_9	s_8

Pattern Nine for All Three Test Sets

M	S_0	S_1	S_2	S_3	C_n	A_0	B_0	A_1	B_1	A_2	B_2	A_3	B_3
s_8	s_7	s_6	s_5	s_4	s_3	s_2	s_1	s_{14}	s_{13}	s_{12}	s_{11}	s_{10}	s_9

Pattern Ten for All Three Test Sets

M	S_0	S_1	S_2	S_3	C_n	A_0	B_0	A_1	B_1	A_2	B_2	A_3	B_3
s_9	s_8	s_7	s_6	s_5	s_4	s_3	s_2	s_1	s_{14}	s_{13}	s_{12}	s_{11}	s_{10}

Pattern Eleven for All Three Test Sets

M	S_0	S_1	S_2	S_3	C_n	A_0	B_0	A_1	B_1	A_2	B_2	A_3	B_3
s_{10}	s_9	s_8	s_7	s_6	s_5	s_4	s_3	s_2	s_1	s_{14}	s_{13}	s_{12}	s_{11}

Pattern Twelve for All Three Test Sets

M	S_0	S_1	S_2	S_3	C_n	A_0	B_0	A_1	B_1	A_2	B_2	A_3	B_3
s_{11}	s_{10}	s_9	s_8	s_7	s_6	s_5	s_4	s_3	s_2	s_1	s_{14}	s_{13}	s_{12}

Pattern Thirteen for All Three Test Sets

M	S_0	S_1	S_2	S_3	C_n	A_0	B_0	A_1	B_1	A_2	B_2	A_3	B_3
s_{12}	s_{11}	s_{10}	s_9	s_8	s_7	s_6	s_5	s_4	s_3	s_2	s_1	s_{14}	s_{13}

Pattern Fourteen for All Three Test Sets

M	S_0	S_1	S_2	S_3	C_n	A_0	B_0	A_1	B_1	A_2	B_2	A_3	B_3
s_{13}	s_{12}	s_{11}	s_{10}	s_9	s_8	s_7	s_6	s_5	s_4	s_3	s_2	s_1	s_{14}

Pattern Fifteen for Test Set One

M	S_0	S_1	S_2	S_3	C_n	A_0	B_0	A_1	B_1	A_2	B_2	A_3	B_3
s_1	s_{12}	s_8	s_{10}	s_{11}	s_5	s_7	s_2	s_{13}	s_4	s_3	s_6	s_{14}	s_9

Pattern Sixteen for Test Set One

M	S_0	S_1	S_2	S_3	C_n	A_0	B_0	A_1	B_1	A_2	B_2	A_3	B_3
s_3	s_{11}	s_8	s_{14}	s_5	s_9	s_{12}	s_2	s_{13}	s_4	s_6	s_{10}	s_7	s_1

Pattern Seventeen for Test Set One

M	S_0	S_1	S_2	S_3	C_n	A_0	B_0	A_1	B_1	A_2	B_2	A_3	B_3
s_3	s_{11}	s_2	s_{14}	s_5	s_9	s_{13}	s_4	s_8	s_4	s_6	s_{10}	s_7	s_1

Pattern Eighteen for All Three Test Sets

M	S_0	S_1	S_2	S_3	C_n	A_0	B_0	A_1	B_1	A_2	B_2	A_3	B_3
s_7	s_6	s_5	s_4	s_3	s_2	s_1	s_8	s_9	s_{10}	s_{11}	s_{12}	s_{13}	s_{14}

Pattern Fifteen for Test Set Two

M	S_0	S_1	S_2	S_3	C_n	A_0	B_0	A_1	B_1	A_2	B_2	A_3	B_3
s_3	s_{11}	s_2	s_{14}	s_5	s_9	s_{12}	s_{13}	s_4	s_8	s_6	s_{10}	s_7	s_1

Pattern Sixteen for Test Set Two

M	S_0	S_1	S_2	S_3	C_n	A_0	B_0	A_1	B_1	A_2	B_2	A_3	B_3
s_8	s_{13}	s_4	s_2	s_{11}	s_1	s_7	s_{10}	s_{14}	s_5	s_6	s_{12}	s_3	s_9

Pattern Fifteen for Test Set Three

M	S_0	S_1	S_2	S_3	C_n	A_0	B_0	A_1	B_1	A_2	B_2	A_3	B_3
s_{13}	s_7	s_3	s_5	s_9	s_{14}	s_{10}	s_6	s_1	s_{11}	s_4	s_{12}	s_8	s_2

Pattern Sixteen for Test Set Three

M	S_0	S_1	S_2	S_3	C_n	A_0	B_0	A_1	B_1	A_2	B_2	A_3	B_3
s_1	s_7	s_{12}	s_8	s_{13}	s_{11}	s_2	s_4	s_9	s_6	s_3	s_{14}	s_{10}	s_5

Pattern Seventeen for Test Set Three

M	S_0	S_1	S_2	S_3	C_n	A_0	B_0	A_1	B_1	A_2	B_2	A_3	B_3
s_{14}	s_7	s_3	s_{11}	s_{13}	s_6	s_2	s_{12}	s_8	s_5	s_3	s_4	s_{10}	s_9

Appendix B

Connection Patterns Used in Section 5.3

Pattern One

A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_0	CI
S_{17}	S_{16}	S_{15}	S_{14}	S_{13}	S_{12}	S_{11}	S_{10}	S_9	S_8	S_7	S_6	S_5	S_4	S_3	S_2	S_1

Pattern Two

A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_0	CI
S_1	S_{17}	S_{16}	S_{15}	S_{14}	S_{13}	S_{12}	S_{11}	S_{10}	S_9	S_8	S_7	S_6	S_5	S_4	S_3	S_2

Pattern Three

A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_0	CI
S_2	S_1	S_{17}	S_{16}	S_{15}	S_{14}	S_{13}	S_{12}	S_{11}	S_{10}	S_9	S_8	S_7	S_6	S_5	S_4	S_3

Pattern Four

A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_0	CI
S_3	S_2	S_1	S_{17}	S_{16}	S_{15}	S_{14}	S_{13}	S_{12}	S_{11}	S_{10}	S_9	S_8	S_7	S_6	S_5	S_4

Pattern Five

A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_0	CI
S_4	S_3	S_2	S_1	S_{17}	S_{16}	S_{15}	S_{14}	S_{13}	S_{12}	S_{11}	S_{10}	S_9	S_8	S_7	S_6	S_5

Pattern Six

A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_0	CI
S_5	S_4	S_3	S_2	S_1	S_{17}	S_{16}	S_{15}	S_{14}	S_{13}	S_{12}	S_{11}	S_{10}	S_9	S_8	S_7	S_6

Pattern Seven

A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_0	CI
S_6	S_5	S_4	S_3	S_2	S_1	S_{17}	S_{16}	S_{15}	S_{14}	S_{13}	S_{12}	S_{11}	S_{10}	S_9	S_8	S_7

Pattern Eight

A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_0	CI
S_7	S_6	S_5	S_4	S_3	S_2	S_1	S_{17}	S_{16}	S_{15}	S_{14}	S_{13}	S_{12}	S_{11}	S_{10}	S_9	S_8

Pattern Nine

A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_0	CI
S_8	S_7	S_6	S_5	S_4	S_3	S_2	S_1	S_{17}	S_{16}	S_{15}	S_{14}	S_{13}	S_{12}	S_{11}	S_{10}	S_9

Pattern Ten

A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_0	CI
S_9	S_8	S_7	S_6	S_5	S_4	S_3	S_2	S_1	S_{17}	S_{16}	S_{15}	S_{14}	S_{13}	S_{12}	S_{11}	S_{10}

Pattern Eleven

A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_0	CI
S_{10}	S_9	S_8	S_7	S_6	S_5	S_4	S_3	S_2	S_1	S_{17}	S_{16}	S_{15}	S_{14}	S_{13}	S_{12}	S_{11}

Pattern Twelve

A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_0	CI
S_{11}	S_{10}	S_9	S_8	S_7	S_6	S_5	S_4	S_3	S_2	S_1	S_{17}	S_{16}	S_{15}	S_{14}	S_{13}	S_{12}

Pattern Thirteen

A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_0	CI
S_{12}	S_{11}	S_{10}	S_9	S_8	S_7	S_6	S_5	S_4	S_3	S_2	S_1	S_{17}	S_{16}	S_{15}	S_{14}	S_{13}

Pattern Fourteen

A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_0	CI
S_{13}	S_{12}	S_{11}	S_{10}	S_9	S_8	S_7	S_6	S_5	S_4	S_3	S_2	S_1	S_{17}	S_{16}	S_{15}	S_{14}

Pattern Fifteen

A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_0	CI
S_{14}	S_{13}	S_{12}	S_{11}	S_{10}	S_9	S_8	S_7	S_6	S_5	S_4	S_3	S_2	S_1	S_{17}	S_{16}	S_{15}

Pattern Sixteen

A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_0	CI
S_{15}	S_{14}	S_{13}	S_{12}	S_{11}	S_{10}	S_9	S_8	S_7	S_6	S_5	S_4	S_3	S_2	S_1	S_{17}	S_{16}

Pattern Seventeen

A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_0	CI
S_{16}	S_{15}	S_{14}	S_{13}	S_{12}	S_{11}	S_{10}	S_9	S_8	S_7	S_6	S_5	S_4	S_3	S_2	S_1	S_{17}

Pattern Eighteen

A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_0	CI
S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9	S_{10}	S_{11}	S_{12}	S_{13}	S_{14}	S_{15}	S_{16}	S_{17}

Pattern Nineteen

A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_0	CI
S_{15}	S_9	S_{13}	S_{14}	S_8	S_6	S_2	S_{11}	S_{17}	S_3	S_{10}	S_{12}	S_4	S_1	S_5	S_{16}	S_7

Pattern Twenty

A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_0	CI
S_4	S_5	S_6	S_7	S_8	S_9	S_{10}	S_{11}	S_{12}	S_{13}	S_{14}	S_{15}	S_{16}	S_{17}	S_1	S_2	S_3

Pattern Twenty One

A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_0	CI
S_8	S_{17}	S_4	S_6	S_9	S_{14}	S_{10}	S_3	S_{13}	S_1	S_{16}	S_{15}	S_2	S_{11}	S_5	S_7	S_{12}

Appendix C

Sequences Generated by LHCA with Rule 239 and LHCA with Rule 22

t	ca 239	ca 22	t	ca 239	ca 22	t	ca 239	ca 22	t	ca 239	ca 22	t	ca 239	ca 22
0	174	11	12	49	23	24	82	52	36	84	4	48	229	27
1	165	17	13	91	51	25	207	38	37	198	14	49	93	41
2	189	58	14	208	45	26	54	25	38	41	29	50	217	6
3	137	59	15	8	8	27	81	46	39	111	32	51	23	9
4	223	57	16	28	20	28	203	13	40	134	16	52	34	22
5	30	62	17	58	54	29	56	24	41	201	56	53	119	49
6	61	53	18	67	33	30	68	44	42	63	60	54	178	42
7	73	36	19	228	18	31	238	10	43	78	50	55	159	3
8	255	30	20	94	63	32	69	19	44	245	47	56	254	5
9	110	37	21	221	55	33	237	61	45	117	15	57	109	12
10	133	28	22	25	35	34	65	48	46	181	31	58	129	26
11	205	34	23	55	21	35	227	40	47	149	39	59	195	43

t	ca 239	ca 22	t	ca 239	ca 22	t	ca 239	ca 22	t	ca 239	ca 22	t	ca 239	ca 22
60	36	1	90	118	46	120	196	12	150	53	52	180	33	42
61	126	2	91	177	13	121	46	26	151	85	38	181	115	3
62	173	7	92	155	24	122	101	43	152	197	25	182	188	5
63	161	11	93	240	44	123	157	1	153	45	46	183	138	12
64	179	17	94	120	10	124	249	2	154	97	13	184	219	26
65	156	58	95	164	19	125	103	7	155	147	24	185	16	43
66	250	59	96	190	61	126	154	11	156	236	44	186	40	1
67	99	57	97	141	48	127	243	17	157	66	10	187	108	2
68	148	62	98	209	40	128	124	58	158	231	19	188	130	7
69	230	53	99	11	4	129	170	59	159	90	61	189	199	11
70	89	36	100	24	14	130	171	57	160	211	48	190	42	17
71	215	30	101	52	29	131	168	62	161	12	40	191	107	58
72	2	37	102	86	32	132	172	53	162	18	4	192	136	59
73	7	28	103	193	16	133	162	36	163	47	14	193	220	57
74	10	34	104	35	56	134	183	30	164	102	29	194	26	62
75	27	23	105	116	60	135	146	37	165	153	32	195	51	53
76	48	51	106	182	50	136	239	28	166	247	16	196	92	36
77	88	45	107	145	47	137	70	34	167	114	56	197	218	30
78	212	8	108	235	15	138	233	23	168	191	60	198	19	37
79	6	20	109	72	31	139	79	51	169	142	50	199	44	28
80	9	54	110	252	39	140	246	45	170	213	47	200	98	34
81	31	33	111	106	27	141	113	8	171	5	15	201	151	23
82	62	18	112	139	41	142	187	20	172	13	31	202	226	51
83	77	63	113	216	6	143	128	54	173	17	39	203	87	45
84	241	55	114	20	9	144	192	33	174	43	27	204	194	8
85	123	35	115	38	22	145	32	18	175	104	41	205	39	20
86	160	21	116	121	49	146	112	63	176	140	6	206	122	54
87	176	52	117	167	42	147	184	55	177	210	9	207	163	33
88	152	38	118	186	3	148	132	35	178	15	22	208	180	18
89	244	25	119	131	5	149	206	21	179	22	49	209	150	63

t	ca 239	ca 22	t	ca 239	ca 22	t	ca 239	ca 22	t	ca 239	ca 22	t	ca 239	ca 22
210	225	55	219	234	44	228	1	32	237	166	27	246	60	12
211	83	35	220	75	10	229	3	16	238	185	41	247	74	26
212	204	21	221	248	19	230	4	56	239	135	6	248	251	43
213	50	52	222	100	61	231	14	60	240	202	9	249	96	1
214	95	38	223	158	48	232	21	50	241	59	22	250	144	2
215	222	25	224	253	40	233	37	47	242	64	49	251	232	7
216	29	46	225	105	4	234	125	15	243	224	42	252	76	11
217	57	13	226	143	14	235	169	31	244	80	3	253	242	17
218	71	24	227	214	29	236	175	39	245	200	5	254	127	58

VITA

Surname: Tian — Give Name: Wanning

Place of Birth: Beijing, China — Date of Birth: Nov 11, 1969

Educational Institutions Attended:

University of Victoria	1993 to 1995
Tsinghua University	1988 to 1990

Degrees Awarded:

B.Sc	University of Victoria	1995

Honours and Awards:

University of Victoria Teaching Award	1995 -1997

PARTIAL COPYRIGHT LICENSE

I hereby grant the right to lend my thesis to users of the University of Victoria Library, and to make single copies only for such users or in response to a request from the Library of any other university, or similar institution, on its behalf or for one of its users I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by me or a member of the University designated by me It is understood that copying of this thesis for financial gain shall not be allowed without my written permission

Title of Thesis

Test Set Generation Using Linear Finite State Machines

Author [redacted]

Wanning Tian

August 7, 1997